Talking to the Semantic Web – Natural Language Query Interfaces for Casual End-users

Doctoral Thesis
for the Degree of a Doctor in Informatics

at the
Faculty of Economics, Business Administration
and Information Technology
of the
University of Zurich

by
Esther Kaufmann

from
Schaenis-Ruettiberg SG, Switzerland

Accepted on the Recommendation of
Prof. Abraham Bernstein, Ph.D.
Prof. Dr. Michael Hess

University of Zurich

February 2009

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, December 5, 2007

The Vice Dean of the Academic Program in Informatics:
Prof. Dr. Gerhard Schwabe

Abstract

The Semantic Web presents the vision of a distributed, dynamically growing knowledge base founded on formal logic. This formal framework facilitates precise and effective querying in order to manage information-seeking tasks. Casual end-users, however, are typically overwhelmed by the formal logic. So how can we help users to query a Web of logic that they do not seem to understand? One solution to address this problem is the use of natural language for query specification, but the development of natural language interfaces requires computationally and conceptually intensive algorithms relying on large quantities of domain-dependent background knowledge, thereby making them virtually unadaptable to new domains and applications, or achievable only by sacrificing retrieval performance. Furthermore, while natural language interfaces hide prohibitive formal query languages, users should know their capabilities in order to utilize the natural language interface successfully.

This thesis proposes to break the dichotomy between full natural language and formal approaches by regarding them as ends of a *Formality Continuum*, where the freedom of full natural languages and the structuredness of formal query languages lie at the ends of the continuum. We hypothesize that portable natural language interfaces to the Semantic Web with high retrieval performance can be built, thereby avoiding a complex configuration by controlling the query language and by extracting the necessary underlying frameworks from ontology-based knowledge bases, since such knowledge bases offer a rich source of semantically annotated information. We further hypothesize that query interfaces for the casual user should impose some structure on the user's input in order to guide the entry, but should not overly restrict the user with an excessively formalistic language. In this way, the best solutions for casual end-users lie somewhere between the freedom of a full natural language and the structuredness of a formal query language.

To support our proposition we introduce, in a first step, four different, domain-independent query interfaces to the Semantic Web that lie at different positions of the Formality Continuum: *NLP-Reduce*, *Querix*, *Ginseng*, and *Semantic Crystal*. The first two

interfaces allow users to pose questions in almost full or slightly controlled English. The third interface offers query formulation in a controlled language akin to English. The last interface belongs to the formal approaches, as it exhibits a formal, but graphically displayed query language. The interfaces are simple in configuration, but still offer well-performing and appropriate tools for composing queries to ontology-based data for casual end-users.

As a second step, we present two evaluations to test our hypotheses: (1) an in-depth retrieval performance evaluation with three test sets comparing our interfaces with three existing systems and (2) a comprehensive usability study with real-world end-users assessing our interfaces and, in particular, their query languages. The two evaluations provide sufficient evidence to determine the advantages and disadvantages of query interfaces at various points along the Formality Continuum. In turn, they lead to concrete answers to our hypotheses.

The thesis shows that natural language interfaces provide a convenient as well as reliable means of querying access to the Semantic Web and, hence, a realistic potential for bridging the gap between the formal logic of the Semantic Web and the casual end-users. As such, its overarching contribution is one step towards the theoretical vision of the Semantic Web becoming reality.

Zusammenfassung

Die Vision des “Semantic Web” ist die Entwicklung einer zusätzlichen semantischen Metaebene zum bestehenden World Wide Web, sodass eine verteilte, dynamisch wachsende Wissensbasis entsteht. Zu diesem Zweck ist das Semantic Web auf einem formalen, logik-basierten Grundgerüst aufgebaut, das eine präzise und effiziente Suche nach Informationen ermöglicht. Gelegentliche oder nicht in Logik geschulte Endbenutzer sind jedoch typischerweise mit Logik überfordert; sie sind zum Teil sogar ausserstande, die formalen logischen Konzepte zu beherrschen. Die grundlegende Frage ist also: Wie können wir den realen Endbenutzern helfen, Informationen in einem semantischen Web zu finden, das sie nicht verstehen? Eine Möglichkeit, diese Lücke zu schliessen, ist die Verwendung von natürlichsprachlichen Zugangs- oder Abfragesystemen. Ein solches natürlichsprachliches Interface erlaubt dem Benutzer den Zugang zu einer Datenbasis, indem eine Anfrage in natürlicher Sprache formuliert wird. Jedoch erfordern leistungsfähige natürlichsprachliche Abfragesysteme rechenintensive Algorithmen und einen immensen Gebrauch von Hintergrundwissen, was zu einem sehr hohen Grad an Domänenabhängigkeit und zu einer Nicht-Portabilität der Systeme führt. Auf der anderen Seite erlauben natürlichsprachliche Interfaces den Endbenutzern, auf eine vertraute und natürliche Weise nach Informationen suchen. Trotzdem kann ein solches Abfragesystem nur effizient genutzt werden, wenn der Endbenutzer weiss, welche Fragen gestellt werden können, da auch ein natürlichsprachliches Suchsystem präzise formulierte Anfragen verlangt, damit diese adäquat verarbeitet und beantwortet werden können.

Diese Dissertation fordert, dass die Dichotomie zwischen formalen, logik-basierten Abfragesprachen und natürlichen Abfragesprachen aufgebrochen wird. Dabei werden die Freiheit/Natürlichkeit von natürlichen Abfragesprachen und die Formalität/Strukturiertheit von formalen Abfragesprachen als Enden eines Formalitätskontinuums aufgefasst. Auf der Basis dieses Formalitätskontinuums werden zwei Hypothesen aufgestellt: (1) Das Portabilitätsproblem von natürlichsprachlichen Interfaces, die am natürlichen Ende des Kontinuums liegen, kann bewältigt werden, ohne dass komplexe, aufwändige

Algorithmen und zeitraubende Implementationsefforts erbracht werden müssen, indem das nötige Wissen, um den natürlichsprachlichen Input adäquat verarbeiten zu können, automatisch aus semantik-reichen Wissensbasen, wie sie das Semantic Web bietet, extrahiert wird. (2) Die zweite Hypothese besagt, dass natürlichsprachliche Abfragesysteme den Endbenutzern bei der Frageformulierung eine gewisse Kontrollstruktur und Einschränkung vorgeben sollen, damit diese durch den Prozess der Anfrageformulierung geführt werden können. Doch die Abfragesprache soll nicht zu formal und strukturiert sein, damit der "normale" Benutzer nicht abgeschreckt wird. Somit liegen die für den realen Endbenutzer sowie auch aus entwicklungstechnischer Sicht besten Suchsystemlösungen irgendwo in der Mitte des Formalitätskontinuums.

Um die beiden Hypothesen zu testen, wurden vier verschiedene Zugangssysteme zu ontologie-basierten Daten entwickelt, die an verschiedenen Positionen im Formalitätskontinuum liegen: *NLP-Reduce*, *Querix*, *Ginseng* und *Semantic Crystal*. Die ersten beiden Systeme erlauben den Benutzern, ihre Anfragen in voller oder leicht eingeschränkter natürlicher Sprache zu formulieren. Das dritte Interface verlangt eine Anfrageformulierung in einer kontrollierten Sprache, die dem Englischen sehr ähnlich ist. Semantic Crystal gehört zu den formalen Suchsystemen, da es eine formale, wenn auch grafisch dargestellte Abfragesprache aufweist. Alle vier Systeme vermeiden eine komplexe, rechenintensive Konfiguration, bieten dennoch performante und mächtige Werkzeuge für nicht-logik-geschulte Endbenutzer bei der Formulierung von komplexen Anfragen zu ontologie-basierten Daten.

Auf der Basis des Formalitätskontinuums und der vier Systeme werden zwei Evaluationen präsentiert: (1) eine Retrieval-Performanz-Evalution und (2) eine Benutzerstudie. Bei der Performanzevaluation wurden die drei natürlichsprachlichen Suchsysteme NLP-Reduce, Querix und Ginseng mit drei existierenden Konkurrenzinterfaces im Hinblick auf die Qualität der Informationsfindung und Systemportabilität einem Benchmark-Test mit drei Datensätzen unterzogen. Die Benutzerstudie umfasste einen Benchmark-Test mit allen vier Suchsystemen in einem kontrollierten Experiment mit 48 realen Endbenutzern. Diese beiden Evaluationen liefern reichlich Anhaltspunkte über die Vor- und Nachteile von Suchsystemen an verschiedenen Positionen des Formalitätskontinuums. Ferner geben sie konkrete Antworten auf die beiden Hypothesen, wo also auf dem Kontinuum die besten Lösungen aus systemtechnischer Sicht sowie, und vor allem, aus Endbenutzersicht liegen.

Die Dissertation zeigt, dass natürlichsprachliche Abfragesysteme ein komfortables sowie zuverlässiges Hilfsmittel offerieren, und somit einen möglichen Schlüssel bilden, um die Potentiale des Semantic Web einem breiten Endbenutzerpublikum zugänglich zu machen. Das alles überspannende Ziel dieser Dissertation ist letztendlich, die theoretische Vision des Semantic Web einen kleinen Schritt in die Realität umzusetzen.

Acknowledgements

On these perhaps most frequently read pages of my thesis, I would like to say thank you to the many people who have joined me on my way towards finishing this thesis. People who know me will read these acknowledgements with a ;-).

First of all, I would like to thank my advisor, Avi Bernstein, for his extraordinary professional support, for teaching me how to read, write, review, ask questions, and do research, for becoming a friend, for hiring me despite my non-informatics background and for convincing me to stay in such a technical research environment.

I want to deeply thank Michael Hess, my co-advisor, for "deporting" me and continuing to support me even after his "demotion" to official co-advisor. I enjoyed every single conversation and hallway meeting that took place during all these years.

Thanks to PIA and its men Moshe Mresse and Christian Schucan. The lectures and, even more, the coffee meetings before and after them, were highlights. You both have greatly inspired me.

Many thanks to my fellow doctoral students, particularly to Christoph Kiefer for patiently sharing the office and singing with me; you are the very best friend one can have! To Jonas Tappolet, Katharina Reinecke, Michael Dänzer, and all other DDIS members. To Geri Reif for being a friend and accompanying me to an amazing place named Holly Springs, and to all SEAL members. You are a great crowd.

Special thanks to Christoph Bürki, Sabine Klauser, Anne Göhring, Niklas Auerbach, Dirk Frohberg, Sylvia Hubalek, Stefan Höfler, Barbara Schwarz, Eda Gregr, Wenzel Doppler, and Kaspar von Gunten, for not only being invaluable friends, also for ensuring culinary changes and delights in the daily campus food.

Thanks to all the diploma students for their crucial contributions and indispensable implementation support: Lorenz Fischer, Christian Kaiser, Beat Sprenger, Renato Zumstein, Guido Badertscher, Stephan Bögli, June von Bonin, Patrick Brunner, Márton Dobozi, Peter Höltschi, Natallia Martchouk, Ursula D'Onofrio, and Andreas Schneider.

Thanks to all tutors, teaching assistants, and assistant researchers for their tireless commitments when papers were submitted or 500 assignments required grading: Gian

Marco Laube, Joachim Fornallaz, Domenic Benz, Michael Hermann, Martina Vazquez, Alex Cejka, Marcel Camporelli, Matthias Altorfer, Martin Constam, David Holzer, Lukas Keller, Valentina Shcherba, Martin Spöerri, Pascal Suter, Werner Winkelmann, and Marko Volic.

Thanks to Eveline Suter, Corinne Maurer, Zehra Kilit, Lotti Kündig, Katrin Häsler, Beat Rageth, and Rico Solcà for keeping things running and always being helpful colleagues at work.

Thanks to all the people who ever participated in one of my multiple, infamous user experiments. I know the tasks were not easy.

Finally, I want to sincerely and deeply say thank you to Harry and my mom Alice, my sisters Prisca with Hanspeter, Nicolas, and Luca, Edith with Ernst, Simon, and Silja, Rita with Daniel, Moira, and little Elia. Needless to say more.

The benchmark data sets, on which the thesis' evaluation and all examples are based, were kindly supplied by Ray Mooney and his group from the University of Texas at Austin (`http://www.cs.utexas.edu/users/ml/nldata.html`).

The Swiss National Science Foundation has generously supported this thesis (Proposal/Project Number 200021-100149/1).

Esther Kaufmann
Zurich, October 2007

Table of Contents

1

Introduction

Over the last decade, the exponential growth of the World Wide Web (WWW) has not only lead to a tremendously increasing number of Web pages, but also to unpleasant, increasing difficulties for users trying to locate and access the information they require. Users often find themselves lost in huge quantities of irrelevant search results, thereby missing relevant information when they are obliged to browse through dozens of results in order to find the appropriate information. It would, therefore, be extremely helpful if computers could assist users and take over some of the information seeking burden by having machine-processable semantics at their disposal through which they could "understand" like humans understand and utilize natural language Web content.

This is exactly what the Semantic Web intends to establish. The Semantic Web is a vision that the present WWW will eventually include the notion of meaning and become a meta-data rich Web where presently human-readable content will have machine-processable semantics [Berners-Lee et al., 2001]. The ultimate goal is that information and data can be shared as well as reused across applications, enterprises, and community boundaries [W3C World Wide Web Consortium, 2007]. The semantic extension of the existing Web consists of meta-data that describe the semantics of the content of Web pages in such a way that machines can process the content. The semantic meta-data is based on concepts that are defined in *ontologies*, which formally define the concepts of a domain. Emerging from the fields of philosophy and, much later, artificial intelligence, ontologies are founded on logic. The languages of logic have a high expressiveness and their formal semantics is well-understood, which leads to unambiguous meanings of logical statements. Hence, the Semantic Web presents the vision of a dynamically growing knowledge base that should allow users to draw on and combine distributed information sources specified in languages based on formal logic.

Common users, however, are typically overwhelmed by formal logic. People have been shown to manifest problems even with the simplest Boolean expressions; the use of the first-order logic formalism underlying the Semantic Web is thus beyond their understanding. Consider the following findings in the literature:

- Experience in the field of *information retrieval* (which is the science of searching for information in documents, searching for documents themselves, or searching within databases, whether stand-alone databases or networked databases such as the WWW) demonstrates that users are better at understanding graphical query interfaces than simple Boolean queries [Spoerri, 1993].

- As queries from Web search engines reveal, the great majority of users simply do not use advanced search features—for example, logical operators and modifiers [Dittenbach et al., 2003]. And even if they do, they make mistakes in more than 50% of cases [Spink et al., 2001].

- The experiments of Turtle [Turtle, 1994] show that expert users consistently performed better in search tasks when formulating natural language queries rather than Boolean queries for full-text legal, statutes, and administrative materials. The possible outcomes with novice users are even more significant. However, note that there are a number of serious problems in Turtle's natural language system, as it does not always provide the most relevant answers in complex queries. Désert [Désert, 1993], therefore, suggests that, for the best results, users should combine the natural language search with a Boolean search, but she admits that a natural language interface such as the one designed by Turtle can be the easier and more efficient manner of retrieving relevant information.

- Bowen *et al.* even demonstrated that computer science students who are trained in composing queries in a logic-based formalism (SQL in their case) are usually inept at composing correct queries in realistically-sized databases instead of the small toy examples used in database classes [Bowen et al., 2004].

From these results, we can legitimately conclude that there is a gap between the emerging logical underpinnings of the Semantic Web and the average users' ability to understand and command formal logic. The fundamental question is then thus: How can we bridge this gap between the logic-based Semantic Web and real-world users, who are at least ill at ease and, oftentimes, unable to use formal logic concepts? In particular, how can we help users to query a Web of logic that they do not seem to understand?

1.1 Motivation

A commonly proposed solution for addressing the gap between common users and formal, logic-based systems is the use of natural languages for knowledge specification and querying. A *natural language interface* (NLI) is a system that allows users to access information stored in some repository by formulating the request in natural language (e.g., English, German, French, etc.). The capabilities of such NLIs go beyond those of keyword-based retrieval systems [Cimiano et al., 2007]. Typical keyword-based or full-text information retrieval systems are not able to return precise answers to questions, but return instead a set of relevant documents containing the given keywords [Baeza-Yates and Ribeiro-Neto, 1999]. Some NLIs allow the use of full natural language, while others restrict the input to a *sublanguage* by a domain or to a *controlled natural language* by grammar and/or lexicon constraints. NLIs access different information repositories: databases, knowledge bases, or—with the emergence of the Semantic Web—ontologies and ontology-based knowledge bases.

While NLIs conveniently hide the formality of ontologies and query languages from end-users by offering them a very familiar and intuitive method of query formulation, the realization of NLIs involves various problems such as discussed in the following:

1. Due to linguistic variability and ambiguities, for which natural languages are infamous, the development of accurate NLIs is *a highly complicated as well as very complex and time-consuming task* that requires extraordinary design and implementation efforts. Natural language processing (NLP) generally requires computationally and conceptually intensive algorithms that presuppose large amounts of domain-dependent background knowledge, which is, to make things more difficult, costly to produce [Badia, 2007]. Nevertheless, by controlling the query language such that the end-user must either follow it or engage the user in query formulation dialogues that are controlled by the system, we can eliminate linguistic variability [Bechhofer et al., 1999, Bernstein et al., 2005a, Schwitter and Tilbrook, 2004]. Moreover, the semantics contained in ontologies can provide the context needed to overcome ambiguities.

2. NLIs with good retrieval performance (i.e., they find all relevant, and only relevant, information) are often domain- or application-tailored, and thus, for the most part, *neither adaptable nor portable*. Particularly, systems that allow full natural language input are in almost every case restricted to the domain of the queried data repository [Frank et al., 2007], and their adaptation to new data repositories can only be accomplished by lengthy manual reconfiguration [Androutsopoulos et al., 1995]. Even though NLP has made good progress in recent years, much current NLI research relies on techniques more reminiscent of traditional information retrieval

than modern NLP [Mooney, 2004]. The systems that can perform complex semantic interpretation and inference tend to require large amounts of domain-specific knowledge and engineering-intensive algorithms, making the systems barely (if at all) adaptable to other domains and applications. Hence, they have a substantial *adaptivity barrier*. However, if we extract the necessary information for analyzing and processing a user's natural language query from a data repository, NLIs can overcome the barrier and successfully become domain-independent or, at least, easily adaptable to new domains [Androutsopoulos et al., 1995, Cimiano et al., 2007].

Note that we define *portable* based on Grosz *et al.*, where a system is said to be portable or domain-independent if it does not require manual customization to be used in a new domain [Grosz et al., 1987]. In contrast, a system is not portable or domain-dependent if it is tailored to one specific domain (e.g., geography) or requires extensive hand-crafted customization for new domains (e.g., gastronomy).

3. The quality of the retrieval performance (in terms of precision and recall, see Chapter 5) of an NLI is usually directly linked to the portability problem. *The more a system is tailored to a domain, the better its retrieval performance is*. The goal, however, is to build portable and therefore valuable NLIs without sacrificing retrieval quality: end-users would not accept unreliable and inaccurate interfaces.

4. Even if we can provide well-performing and portable NLIs, another problem arises from the users' side. Typically, users do not know what capabilities a natural language system has, since most NLIs do not help their users in assembling queries, and this sometimes leads to a "clean sheet of paper" effect, also known as writer's block [Bell and Rowe, 1992, Chakrabarti, 2004, Odgen and Bernick, 1997] [Tennant et al., 1983]. Consequently, users should be guided or at least supported when building queries [Auer and Lehmann, 2007]. Otherwise, since the user does not know what can be asked, many questions will be misunderstood by an NLI, and may even be rejected if the questions exceed or fall short of the capability of the system. The mismatch between the users' expectations and the capabilities of a natural language system is called the *habitability problem* [Thompson et al., 2005], and causes many users to revert to familiar techniques such as the use of keywords [Dittenbach et al., 2003, Malhotra, 1975, Tennant et al., 1983]. Current NLP tools, while easier to learn than formal logic, still suffer from the habitability problem (as they only understand some subset of natural language), but sometimes suggest full understanding. Moreover, since users type in regular sentences, they are tempted to anthropomorphize, and assume that the computer actually understands their queries. Natural language systems, however, still require carefully developed query statements. Further weaknesses include their general inability to provide

more than limited error messages, and that they oftentimes, due to the imprecise nature of natural language, make the composition of complex queries tedious [Mooney, 2004, Thompson et al., 2005]. Synonyms and ambiguous expressions are, in particular, obstacles. Thus, for the successful use of an NLI, users need to know what is possible to ask [Androutsopoulos et al., 1995, Bechhofer et al., 1999] and which question formulations are valid [Feldman, 1996].

5. These issues are reflected in the literature's repeatedly discussed question of whether NLIs are practical and appropriate compared to formal query languages—however without any conclusive answer [Bell and Rowe, 1992, Chakrabarti, 2004] [Dekleva, 1994, Désert, 1993, Mooney, 2004, Paris and Tibbo, 1998] [Thompson et al., 2005, Turtle, 1994]: Formal query languages have been found inaccessible by casual users, but offer a rich tool for composing complex queries by experts; systems applying natural language query languages are afflicted with the adaptivity barrier and the habitability problem. Results from the studies comparing NLIs and Boolean or logic-based search systems indicate that different users and different queries demand different retrieval mechanisms: natural language searching is comprehensive and effective for vague or broad questions where the user is willing to tolerate less relevant and even unrelated items in the retrieved set. Logic-based searches, in contrast, are precise, and appropriate for expert users with high demands in terms of retrieval quality. For best overall results, searchers need to employ both search techniques [Paris and Tibbo, 1998, Tomaiuolo and Packer, 1998] if they are available.

   Such discussions generally raise the *issue of the usefulness of NILs*. Even if we design a well-performing and portable NLI, it is unclear whether it will be approved and adopted by end-users. In the time of Google and graphical user interfaces, where people are used to formulating their information needs with keywords and then browsing through dozens of answers to find the appropriate one or clicking through menus and graphically displayed functions, interfaces with full natural language input may be utterly redundant.

The domain-dependency of intelligent NLIs and the habitability problem account for the fact that we are still underway in our quest to see the successful use of full natural language to command and query the Semantic Web. We still believe that NLIs are a promising option for casual end-users wishing to interact with the logic-based knowledge bases of the Semantic Web without having to learn formal languages such as logic in order to utilize and benefit from the innovations of the Semantic Web.

To that end, this thesis proposes to break the dichotomy between full natural language approaches and formal, logic-based query approaches by regarding them as ends of a

Formality Continuum where the freedom of a full natural language and the structuredness of a formal query language lie at the ends of the continuum (see Figure 1.1).[1] Based on *structuration theory* [Giddens, 1984, Orlikowski, 1992], which states that structure enables action by providing a guide, but can also constrain when the guide overly constricts expressibility, we argue that query interfaces should impose some structure on the user's input to guide the entry, but not overly control the user with an excessively formalistic language.

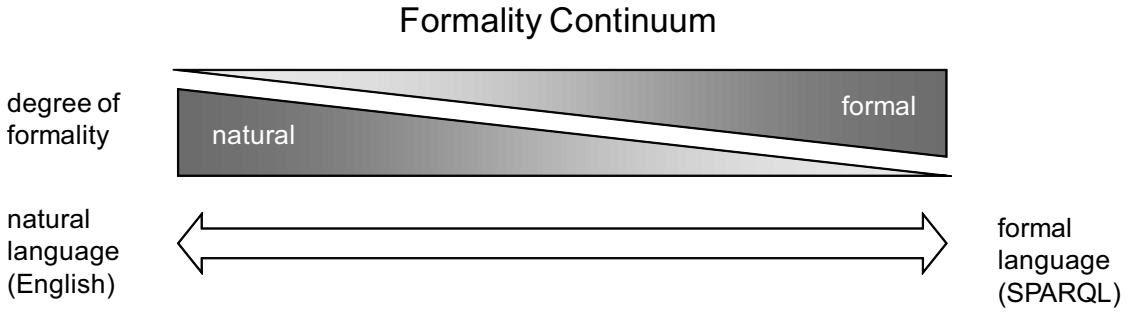


Figure 1.1: The Formality Continuum regards the freedom/naturalness of natural languages and the structuredness/formality of formal query languages as ends of a continuum.

Specifically, we intend to bring full natural language approaches and formal, logic-based approaches closer to each other, since we propose that the best solutions for the casual and occasional end-user (in contrast to expert users) will lie somewhere in the middle of the Formality Continuum, as this provides the best tradeoff between structuredness and freedom, therefore tackling the habitability problem. The current situation is that end-users are situated somewhere between the impreciseness of uninterpreted keyword systems and the rigor of formal query engines. Thus, search systems will either let users express information needs naturally and analyze their queries more intelligently [Chakrabarti, 2004], or will allow enhancements to control the user's search process, as shown by the Formality Continuum. We further base our proposition on experience with controlled natural languages, which have shown that they are much easier to learn for casual end-users than formal languages like logic, and are sufficient for querying structured knowledge bases [Bechhofer et al., 1999, Cha, 1991, Malhotra, 1975, Schwitter and Tilbrook, 2004, Tablan et al., 2005, Thompson et al., 2005].

[1] We are fully aware that the Formality Continuum is a radical simplification of the notions of naturalness and formality, which are constituted of many more objective as well as subjective parameters. It still provides us with an applicable classification system for query interfaces with regard to their query languages and a basis to develop our hypotheses.

When addressing casual end-users, we refer to them as defined by Battle:

> "[Casual end-users are] people who are either seeking information or trying to accomplish something in the course of their everyday life or work. They do not know what the Semantic Web is, and they do not care as long as they can get what they need quickly."
>
> [Battle, 2006]

To support our proposition regarding the best solutions for casual end-users, we present four different query interfaces to the Semantic Web that lie at different positions of the Formality Continuum:

- NLP-Reduce
- Querix
- Ginseng
- Semantic Crystal

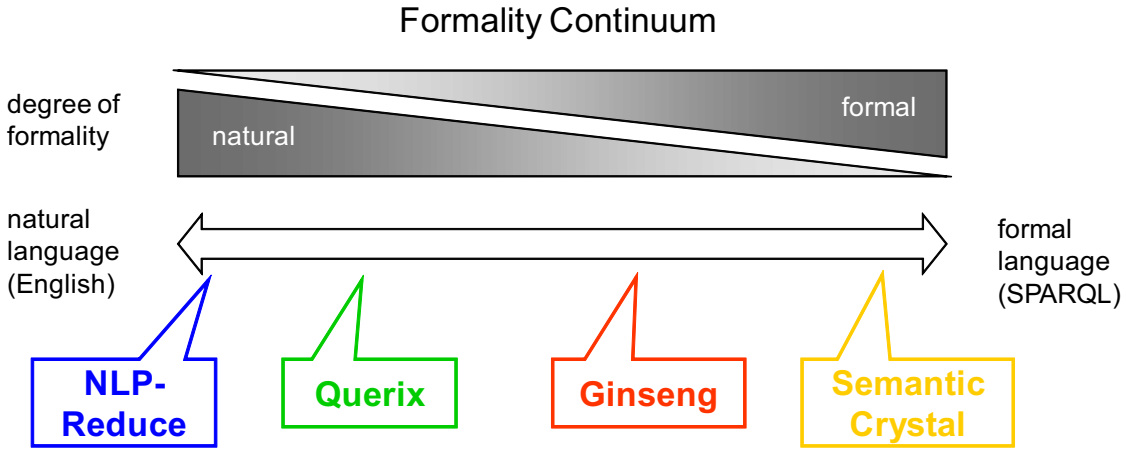


Figure 1.2: The four query interfaces NLP-Reduce, Querix, Ginseng, and Semantic Crystal to query Semantic Web data support different query languages with regard to their degree of freedom, naturalness, structuredness, and formality, therefore providing different positions distributed along the Formality Continuum.

The first two interfaces allow users to pose questions in full or slightly controlled English. The third interface offers query formulation in a controlled language akin to English. These three interfaces are, therefore, considered to be NLIs, and lie on the natural end of the Formality Continuum or towards its middle. The last interface belongs to

the formal approaches, as it exhibits a formal, but graphically displayed query language. Since each of the four interfaces supports a different query language with a different degree of control and formality, they provide different positions along the Formality Continuum, as illustrated in Figure 1.2.

Assuming that we can address the previously-mentioned causes for the habitability problem via a natural query language that is guided and controlled in order to support the user's query formulation tasks, we think that natural language query interfaces offer a real alternative for casual end-users to interact with the Semantic Web and its logic-based knowledge bases. Furthermore, we believe that we can overcome the adaptivity barrier without having to apply complex, knowledge-intensive algorithms and undertake time-consuming implementation efforts by controlling, to some extent, the natural query language, and by extracting the necessary knowledge to process natural language queries from semantically-enriched knowledge bases. These two major issues, which we identified when reviewing the field of NLIs, underlie our thesis' two hypotheses, as presented in the next section.

1.2 Hypotheses

Though we have identified five problem dimensions regarding NLIs—and there may be other—we think that NLIs are a promising option for casual end-users to interact with logic-based knowledge bases [Reif, 2005]. Several projects have shown that NLIs can both perform well in retrieval tasks [Frank et al., 2007, Popescu et al., 2003] [Tang and Mooney, 2001] and be portable [Cimiano et al., 2007, Lopez et al., 2006a] [Wang et al., 2007] without being unnecessarily complex, and can, as such, tackle the adaptivity barrier. Some studies also investigated the usefulness of natural language for different tasks with regard to end-users [Cimiano et al., 2007, Dittenbach et al., 2003, Duke et al., 2007, Jarke et al., 1985, Malhotra, 1975, Reichert et al., 2005], therefore addressing the habitability problem. Their findings provide important insights, but do not provide a conclusive answer.

Following the nomenclature of the two basic problems of NLIs, our thesis' two hypotheses are the following:

1. Adaptivity Hypothesis

   Our first hypothesis is called the *Adaptivity Hypothesis* and proposes that portable and well-performing NLIs for the Semantic Web can be built by both controlling the natural query language to some extent and by extracting the information necessary for processing natural language queries from the underlying ontology-based knowledge bases. Such knowledge bases, being semantically annotated, offer a

richer source of information than, for example, databases or unstructured natural language documents. Specifically, we propose that the complexity of such a query system can be reduced to a reasonable complexity and development effort for the creator.

The key to this hypothesis is a combination of "simplicity" and "semantics," meaning that the complexity of NLIs can be drastically reduced (when comparing it to the usual full NLIs) because the semantics contained in ontology-based data repositories compensates for the reduction, thereby still maintaining a high quality of retrieval performance. Simplicity is achieved by asserting some control over the input language.

2. Habitability Hypothesis

   The second hypothesis is called the *Habitability Hypothesis*. It proposes that query interfaces to the Semantic Web should impose some structure on the casual end-user in order to guide the query formulation process without overly restricting the user with an excessively formalistic language, and thereby alienating him or her. As such, the best solutions for casual or occasional end-users lie somewhere in the middle of the Formality Continuum, therefore easing the query formulation task.

   The key to the habitability hypothesis is "asking the user." Our underlying premise is that NLIs are only useful for casual end-users if they are actually approved and, therefore, used by them. The evaluation of the hypothesis will comprise a comprehensive usability study with real-world users in order to discover which query languages they like to use. Furthermore, it will throw some light in general on the problem dimension of usability and usefulness of NLIs (i.e., the last of the five issues raised above).

To evaluate the two hypotheses, we performed the following two steps:

1. We developed and implemented a total of four different query interfaces for the casual and occasional end-users to query Semantic Web knowledge bases, and conducted a thorough test set evaluation of the systems according to established methodologies showing the retrieval performance of the systems with three data sets from three different domains and in comparison with three existing NLIs. This evaluation allowed us to generate conclusive evidence regarding the adaptivity hypothesis.

2. In a second step we conducted a comprehensive usability study by benchmarking the tools against each other in a controlled experiment. The goal was that casual end-users should test and assess the usability of each of the four systems and, in

particular, their query languages. This provided us with sufficient evidence to determine the advantages and disadvantages of query interfaces at various points along the Formality Continuum. In turn, this lead to concrete answers to the question of where on the Formality Continuum the best query interface solutions for the casual end-user lie, as such addressing the habitability hypothesis.

The overarching goal of the thesis is to push the vision of the Semantic Web further into realization and to provide a chance to offer the Semantic Web's capabilities to the general public, which can only happen if we bridge the gap between the real-world users and the logic-based scaffolding of the Semantic Web.

1.3 Contributions

While moving along the development and evaluation of the hypotheses, our thesis makes both practical as well as scientific contributions. The practical contributions are the following:

- As long as the Semantic Web is not accessible and, therefore, not useful to a wide population of users, but only to a few researchers, it will remain a theoretical vision and not become reality. Therefore, the thesis' findings with regard to the tools, the concept of the Formality Continuum, and the additional insights gained from the evaluations provide a substantial contribution for future Semantic Web applications.
- Furthermore, the interfaces we developed seem to have the potential to facilitate convenient query interfaces for real-world applications and have already attracted attention from people with real ontology-based data and the urgent need to provide query access to the data. The interfaces could supply a wide range of users with query interfaces that support their information searches.

Our thesis also presents a variety of scientific contributions, namely:

- By making Semantic Web ontologies accessible to casual users, our thesis advances research in the Semantic Web enabling the widespread use of the theories and applications developed in this quickly expanding research field.
- The thesis also looks at how much syntactic and semantic analysis is needed when processing natural language questions posed to structured knowledge bases. It, hence, investigates whether a full NLP machinery is required for these tasks, or

whether a limited set of NLP techniques provide sufficient results, which would have lower computational requirements and require less manual adaptation—consequently addressing the adaptivity barrier.

- By providing query interfaces at different points at the Formality Continuum evaluating the habitability hypothesis, general insights into the field of query languages ranging from graphical, guided, and controlled to full natural language query languages can be acquired. This contributes to the scientific discussion of whether natural language query languages are in fact useful and which level of natural language is best for common users.

- Finally, our evaluation will consist not only of a retrieval performance evaluation, as most other studies do, but also of a thorough and comprehensive usability study with real-world people. Consequently, the results gained in this study contribute an important corner-stone to the discussion of the usefulness and usability of natural language query interfaces for the general public.

1.4 Structure of the Thesis

The remainder of the thesis is structured as follows:

Chapter 2 introduces the vision of the Semantic Web and presents its basic concepts, technologies, and frameworks. In doing so, the chapter also establishes the major terminology of the Semantic Web research field.

Chapter 3 contains the review of the related work that influenced the development and evaluation of our interfaces in particular and our work in general. The projects and their literature are divided into the three categories: natural language interfaces to databases, natural language interfaces to Semantic Web knowledge bases, and guided natural language interfaces. Furthermore, the review specifically focuses on NLI projects that include usability studies.

In Chapter 4, we present each of the four interfaces and explain their major characteristics along the Formality Continuum, beginning with that possessing the least controlling and most natural query language, then continuing with the systems that feature more controlling query languages, and closing with the system requiring a graphical formal query language. For each system, we will show how a user experiences query formulation, present the technical setup, and discuss more details of the functionality.

Chapter 5 presents the experimental setup, the methodology, and the results of the retrieval performance evaluation, where our interfaces were compared with three exist-

ing NLIs with regard to portability and retrieval performance on three different test data sets from three domains.

Chapter 6 then describes the usability study, in which the four systems are benchmarked against each other in a controlled experiment with 48 real-world users, and discusses the results.

Chapter 7 presents a discussion of some limitations to the design and implementation of the four systems as well as the conduction of the two evaluations and the overall approach, which leads to possibilities for future work and enhancements.

Finally, Chapter 8 summarizes the most relevant findings and closes with general conclusions.

Before closing the first chapter, we need to clarify the use of the terms *query* and *question*. In fact, we do not distinguish between the two terms, and use them interchangeably—a method that contrasts with what is often the literature's common usage. Badia, for example, reserves the word *query* for formal queries such as database queries that formalize a user's information request, and the word *question* for the request as posed by the user in natural language [Badia, 2007]. In our case, the division cannot be so sharply drawn. For example, in one of our query interfaces (i.e., Ginseng) the question composed by a user looks like a normal natural language question, while for the system, since the question is incrementally and synchronously translated into a formal query during question formulation, it is already a query. Another system, Semantic Crystal, does not even incorporate natural language questions, but merely formal queries. Furthermore, we use the terms synonymously, since a natural language query can either be expressed as a real question or interrogative sentence (e.g., "What is the height of Mount Whitney?"), as an imperative sentence (e.g., "Give me the height of Mount Whitney!"), as a sentence fragment (e.g., "height of Mount Whitney"), or even consist of just keywords (e.g., "high Mount Whitney"). As such, since it is not always a proper question, we want to be able to refer to a query that a user poses by more than just *question*.

2

The Vision of the Semantic Web

The cornerstone of the Semantic Web was laid in 1998 by Tim Berners-Lee, James Hendler and Ora Lassila [Berners-Lee et al., 2001]. The Semantic Web is the compelling vision that the present WWW will include the notion of meaning and become a meta-data rich Web where presently human-readable content will have machine-processable semantics. The goal is that information and data can be shared as well as reused across applications, enterprises, and community boundaries [W3C World Wide Web Consortium, 2007].

Even though the present Internet provides an extremely simple way to share information, the ability to read, understand, and process content is restricted only to humans. Computers have great difficulties processing these natural language documents, let alone processing them automatically. However, information desired by humans is also hard to find, as the precision of search results is low. Current Web search engines such as Google and Yahoo perform efficient but imprecise searches, thereby burdening users with thousands of answers to a single query. Users must thus assess their results by browsing and deciding for themselves what is relevant or not. The result is an immense overload of information for users.

Today's Web search engines do not help interpret search results. It would, however, be of great help if computers could assist users and alleviate some of the burden by having machine-processable semantics at their disposal in order to understand" results like a human and thereby utilize natural language Web content. This is exactly what the Semantic Web is thought to establish. Berners-Lee and his colleagues sketch the idea as follows:

> "It is an extension of the current Web in which information is given well-defined meaning, better enabling computer and people to work in cooperation."
>
> [Berners-Lee et al., 2001]

The extension of the existing Web consists of meta-data that describe the semantics of the content of Web pages in such a way that machines can process the content. The semantic meta-data is based on concepts that are defined in *ontologies*. An ontology formally defines the concepts of a domain. It furthermore defines the properties of the concepts as well as the relationships that exist between concepts. Breitman *et al.* give the following definition of ontologies, which we will adopt in this thesis:

> "Ontologies are conceptual models that capture and make explicit the vocabulary used in a domain or in a semantic application, thereby guaranteeing the absence of ambiguities."
>
> [Breitman et al., 2007]

Gruber adds to this definition as follows:

> "An ontology is readable for both humans and machines. Together with a syntax and semantics, it provides the language by which knowledge-based systems can interoperate, e.g. exchanging assertions, queries, and answers. The ontology determines what "exists" for a system."
>
> [Gruber, 1992]

Consider, for example, the domain of geography. In this domain, there are concepts such as "country," "city," "capital," "mountain," "lake," etc. Each "city" has a property "has population" and a relationship "is in country;" the latter links the "city" to the concept "country." We can say that the ontology provides concepts, properties, and relationships with well-defined meanings such that the geographical information of Web pages can be described and annotated by using the elements of the ontology.

Ontologies have been a research topic addressed in the fields of knowledge modeling and knowledge representation [Staab and Studer, 2004]. Knowledge representation, in turn, was studied long before the emergence of the WWW in the area of artificial intelligence and, a long time before that, in logic or philosophy [Antoniou and Hermelen, 2004]. It is not surprising that logic is still the foundation of ontologies and, hence, the whole Semantic Web, since logic is the foundation of knowledge representation, particularly in the form of predicate logic or first-order logic. The languages of logic have high expressiveness, and their formal semantics is well-understood: this leads to unambiguous meanings of logical statements. There are sound and complete proof systems to derive new statements from premises optimal for the Semantic Web.

The research community that has recently emerged around the Semantic Web has developed and published a framework comprising a variety of languages, and standard

recommendations based on the logical groundwork of knowledge representation. In particular, the *World Wide Web Consortium* (W3C) wants to bring the vision of the Semantic Web into realization. The W3C is an international consortium where member organizations, a full-time staff, and the public work together to develop Web standards [W3C World Wide Web Consortium, 2007]. Its mission is to lead the WWW to its full potential by developing protocols and guidelines that ensure long-term growth. A number of specification recommendations have been published for the Semantic Web. The role of W3C in their promulgation is to draw attention to these specifications and to promote their widespread deployment.

In the following sections, we will describe the major specification recommendations, namely the *Resource Description Framework* (RDF), the *Resource Description Framework Schema* (RDFS), the ontology language *OWL*, and the query language *SPARQL*, as these specifications form the basic concepts and the building blocks of the present state of the Semantic Web and its applications. In doing so, we will also introduce the basic terminology related to the Semantic Web and its models, and will present some Semantic Web tools (e.g., Jena, Sesame, Pellet, Protégé, etc.) that have been developed and possess general approval, and therefore contribute substantially to the propagation of the Semantic Web.

2.1 Resource Description Framework (RDF)

The *Resource Description Framework* (RDF) is a general purpose language for representing information in the Web in a minimally constraining and maximally flexible way [Klyne and Carroll, 2004]. Its purpose is the processing of machine-processable information without loss of information. RDF is intended for situations where information needs to be processed and exchanged between applications rather than only being displayed to people [Manola and Miller, 2004].

In the Semantic Web, RDF is the data-model for representing the meta-data [Antoniou and Hermelen, 2004, Klyne and Carroll, 2004]. It provides users with a domain-independent framework for representing information about resources in the WWW. With it, one can unambiguously express and formalize the meaning of concepts and facts. Everything that can be described is called a *resource*. A resource can be anything: a city, a book, a person, a process, a company, etc. Resources being described have properties which have values. These properties and values are specified by describing the resources in *RDF statements*. The part that identifies the resource the statement is about is called the *subject*. The part that identifies the property of the subject that the statement specifies is the *predicate*, and the part that identifies the value of that property is the *object*. Hence, an *RDF statement* is a triple consisting of a subject, a property, and an object.

The subject is what we make a statement about; say, for example, if we wanted to make a statement about a *city*. A property is a special kind of resource; it defines what kind of information is stated about the subject. Our statement could thus regard the *population size* of a city. Finally, an object defines the value of a property. In our example, it would provide the *population figure* of a specific city.

An RDF triple asserts that some relationship, indicated by the predicate, exists between the things denoted by subject and object of the triple. RDF properties may be thought of as attributes of resources, and correspond to traditional attribute-value pairs; RDF properties also represent relationships between resources. The triple depicted below expresses in *triple notation* that the resource "Atlanta" has a relationship to the resource "Georgia" and that the meaning of the relationship is "is capital of".

```
Atlanta isCapitalof "Georgia" .
```

The object of an RDF statement is either a *literal* or other resource. Literals are character strings (i.e., strings, integers) that represent atomic values such as numbers and dates or strings such as "Georgia" in the above example. They are not used as subjects or properties in RDF statements.

In RDF, each resource is identified by a *Uniform Resource Identifier* (URI) [Manola and Miller, 2004]. If the resource is identified on a Web page, the URI is a Web address: a *Uniform Resource Locator* (URL). The character string URI unambiguously identifies a resource though it does not necessarily provide Web access to the resource. Consider the following triple with typical notation including the URLs:

```
<http://www.ifi.uzh.ch/ddis/ont/nli/geo#Capital>
<http://www.ifi.uzh.ch/ddis/ont/nli/geo#isCapitalOf>
<http://www.ifi.uzh.ch/ddis/ont/nli/geo#State> .
```

For convenience and readability, an abbreviated form is mostly used to represent URI references. A name in the form of `prefix:suffix` can be interpreted as a URI reference consisting of the URI associated with the prefix concatenated with the suffix. So, for example, if the prefix `geo` is assigned to the namespace URI `http://www.ifi.uzh.ch/ddis/ont/nli/geo.owl#` by defining the assignment in the ontology model as follows,

```
<rdf:RDF
   xmlns:geo = "http://www.ifi.uzh.ch/ddis/ont/nli/geo.owl#"
>
```

then `geo:Capital` is shorthand for the URI reference

```
http://www.ifi.uzh.ch/ddis/ont/nli/geo#Capital.
```

The example of above exhibiting complete URIs is shown in abbreviated form here:

```
geo:Capital  geo:isCapitalOf  geo:State .
```

URI references are used for naming many things in RDF. The *namespace*, which gives the context of resources, is a collection of names identified by a URI. Names of namespaces cannot have more than one meaning. This supplies a global, worldwide, and unique naming scheme.

The following example uses the namespace prefix `rdf` assuming the binding of the prefix to the RDF specification `http://www.w3.org/1999/02/22-rdf-syntax-ns#`:

```
geo:Atlanta  rdf:hasType  rdf:XMLLiteral .
```

Triple statements in RDF represent a directed, labeled graph where subjects and objects are the nodes and predicates constitute the arcs between nodes. Resources are typically depicted as ellipses, and literals as rectangles. The examples

```
geo:Capital  geo:isCapitolOf  geo:State .
```

and

```
<http://www.ifi.uzh.ch/ddis/ont/nli/Springfield>
   geo:isCapitalof  "Illinois" .
```

are represented as graphs in Figures 2.1 and 2.2 respectively, where the first example is shown with abbreviated URIs.

Figure 2.1: The triple [`geo:Capital geo:isCapitolOf geo:State.`] represented as graph with abbreviated URIs to increase readability.

Besides the triple notation, RDF also provides an XML-based syntax called *RDF/XML* for recording and exchanging RDF graphs. It is a normative syntax for RDF defined in

[Beckett, 2004]. Based on the RDF vocabulary and a set of URIs, it is usually preferred, in the Semantic Web community, to triple notation. Figure 2.3 shows the example shown as graph in Figure 2.2 in RDF/XML notation.

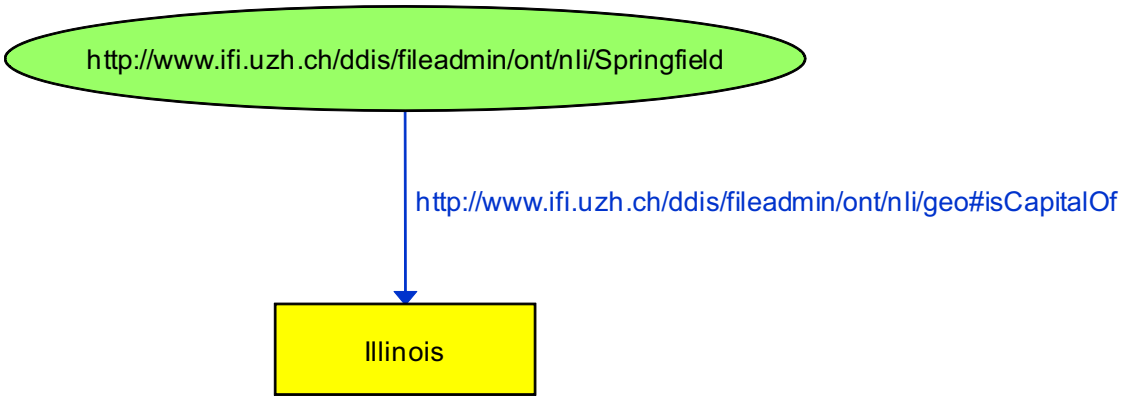


Figure 2.2: The triple [`"Springfield" geo:isCapitalof "Illinois".`] represented as graph with typical ellipses used for resources and rectangles for literals.

RDF has an abstract syntax that reflects a simple graph-based data model and a formal semantics with a rigorously defined notion of entailment providing a basis for well-founded reasoning such as deductions about the meaning of expressions in RDF data [Klyne and Carroll, 2004]. As such, it supports the notion of entailment and the rules of inference. Consider, for example, that the resource "Capital" is defined as a subclass of "City" (for the explanation of the subclass concept see Chapter 2.2). If "Atlanta" is asserted to be the capital of Georgia, we can then infer that "Atlanta" is not only a capital but also a city. We will further review the concept of *reasoning* with a focus on tools to process the mechanism in Chapter 2.5.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:geo = "http://www.ifi.uzh.ch/ddis/fileadmin/ont/nli/geo#"
         xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    <rdf:Description rdf:about =
        "http://www.ifi.uzh.ch/ddis/fileadmin/ont/nli/Springfield"
        <geo:isCapitalOf>Illinois</geo:isCapitalOf>
    </rdf:Description>

</rdf:RDF>
```

Figure 2.3: The triple [`"Springfield" geo:isCapitalof "Illinois".`] represented in RFD/XML notation.

2.2 RDF Schema (RDFS)

While RDF is domain-independent and extremely flexible, it provides no means for defining specific domains or applications. The *RDF Schema* (RDFS) aims at defeating this lack of expressiveness. RDFS is an ontology definition language with which one can define a vocabulary in order to describe the resources of a domain via meta-data [Brickley and Guha, 2004]. In other words, it is an ontology language offering certain modeling primitives with a fixed meaning. As such, RDFS is a semantic extension of RDF for defining the vocabulary of an ontology by describing groups of related resources (i.e., classes) and the relationships between these resources (i.e., properties). It offers primitives to model hierarchies of such classes and properties. The descriptions are based on RDF triple format.

The term *RDF Schema* might be misleading for readers familiar with the XML Schema, since the relation between the XML Schema and XML is not the same as between the RDF Schema and RDF. While the XML Schema constrains the structure of an XML document, the RDF Schema defines the controlled vocabulary of an RDF data-model.

Similar to the RDF specification, each term defined in RDFS holds the XML namespace prefix `rdfs`. This namespace is identified by the URI reference `http://www.w3.org/2000/01/rdf-schema#`. There are currently 13 RDF classes, such as `rdfs:Resource`, `rdfs:Class`, `rdfs:Datatype`, and `rdfs:Literal`, defined for RDFS on W3C [Brickley and Guha, 2004]. As well, there are 16 RDF properties defined, for example `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`, `rdfs:label`.

A class is any resource having an `rdf:type` property whose value is a resource from the RDF Schema vocabulary `rdfs:Class`. The members of a class are known as *instances* of a class, sometimes also called *individuals*. The properties again define relations between subject resources and object resources. They can demand type restrictions of their subject and values. If a restriction is imposed on the subject, the *domain* of the property is restricted, while if we impose a restriction on the values of a property, the *range* of the property is restricted. A triple of the form

```
geo:runsThrough rdfs:domain geo:River .
```

states that `geo:runsThrough` is an instance of the class `rdfs:property`, that `geo:River` is an instance of the class `rdfs:Class`, and that the resources denoted by the subjects of triples whose predicate is `geo:runsThrough` are instances of class `geo:River`. Verbalized, this ensures that only "rivers" can have the property "runs through."

Analogously, the triple

```
geo:runsThrough  rdfs:range  geo:State .
```

states that `geo:runsThrough` is an instance of the class `rdfs:property`, that `geo:State` is an instance of the class `rdfs:Class` and that the resources denoted by the objects of the triples whose predicate is `geo:runsThrough` are instances of the class `geo:State`. The specification of the range ensures that only "states" can be in object position of the property "runs through."

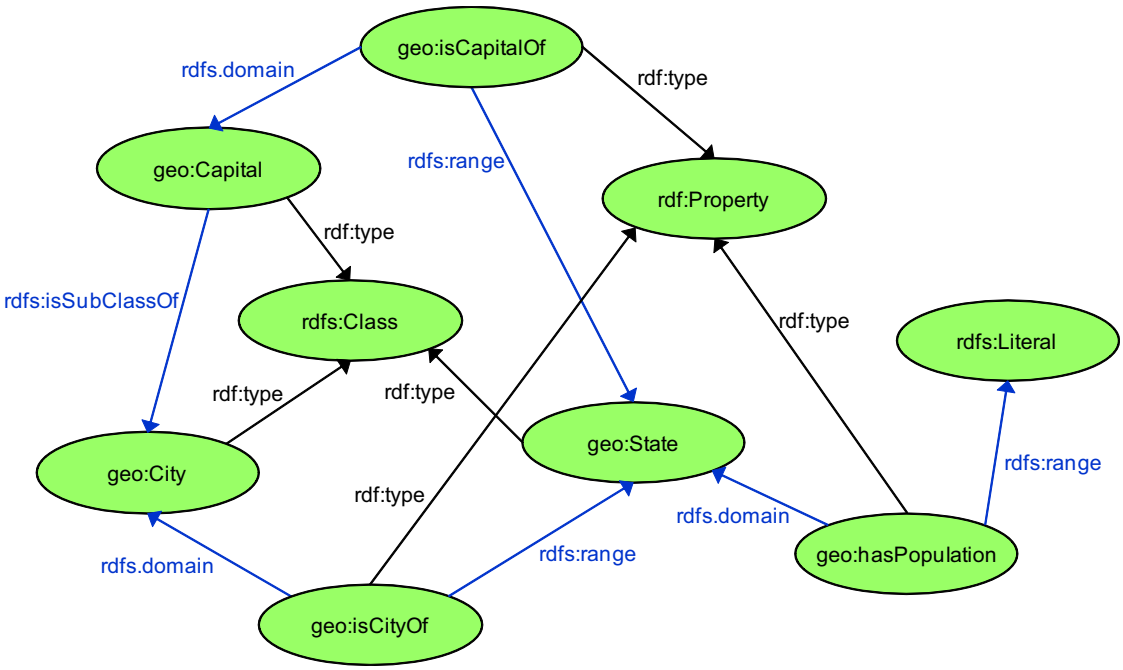


Figure 2.4: Graph representation of an example ontology for geographic information including the RDFS primitives `rdfs:domain`, `rdfs:range`, and `rdfs:isSubClassOf` depicted in blue.

As RDFS uses the RDF triple format, it can, therefore, also be represented as a graph where classes are the nodes and properties the arcs between them. Figure 2.4 depicts a small example ontology that encodes geographic information. The ontology contains three classes (i.e., `geo:Capital`, `geo:City`, `geo:State`) which are identified as such through the RDF type definition `rdf:type`. The class `geo:Capital` is additionally declared as a subclass of the class `geo:City`. There are also three ontology-specific properties (i.e., `isCityOf`, `isCapitalOf`, `hasPopulation`) each holding domain and range specifications. The range of the property `hasPopulation` is specified as `rdfs:Literal`, so the property's object cannot be an instance of the three classes from the namespace `geo`, but must possess a literal value. RDF properties are depicted in black,

RDFS properties in blue.

The RDF Semantics specification mentioned in the last section also provides a precise semantics and corresponding systems of inference rules for RDFS giving its elements well-defined meanings and offering rules for reasoning [Hayes, 2004].

We now want to clarify some terminology. It is customary for *ontology* to refer only to the *ontology schema*, also known as the *ontology model* or *ontology meta-model*. An ontology, hence, is merely a specification of conceptualization without naming instances. If instances are annotated by ontology tags and modeled as ontology, then we speak of a *knowledge base* or an *ontology-based knowledge base*. Thus, a knowledge base is a collection of instances of the concepts defined in the ontology, and the ontology provides the structure of the knowledge stored in the knowledge base.

Further terminology includes the terms *Abox* and *Tbox*, which are used to describe the two different statement types in ontologies and knowledge bases. Tbox statements describe the controlled vocabulary or the set of classes and properties of an ontology. Abox statements are Tbox-compliant statements regarding the vocabulary that describe instances. Together, all Abox and Tbox statements make up the ontology-based knowledge base.

2.3 Web Ontology Language OWL

The limited expressiveness of RDFs resulted in the need for a more powerful ontology modeling language—in particular, one that permitted greater machine interpretability of Web content. This lead to the W3C recommendation of the *Web Ontology Language OWL* [McGuinness and Harmelen, 2004]. OWL allows modelers to use an expressive formalism to define various logical concepts, and relations in ontologies to annotate Web content. The enriched content can then be consumed by machines in order to assist humans in various tasks. As such, OWL fulfils the requirement for an ontology language that can formally describe the meaning of terminology on Web pages. If machines and applications are expected to perform useful reasoning tasks on these Web documents, the language must surpass the semantics of RDFS. OWL has been designed to meet this need.

Similar to RDFS, OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. The resulting ontologies are used by applications that need to process the content of information instead of just presenting it. OWL provides a richer vocabulary along with formal semantics than RDFS by allowing additional modeling primitives that result in an increased expressivity for describing properties and classes. The following shortcomings of RDFS, as well as others not listed, are covered in OWL:

Equality of Classes. The declaration `owl:equivalentClass` can be used to indicate that two classes have precisely the same instances, and `owl:sameAs` is defined between two classes to indicate that they are identical in every way. Such class declarations are often used if two ontologies need to be unified. We could, for example, state that the class `Capital` is equivalent to the class `Capital` in another ontology on geography:

```
<owl:Class rdf:ID="Capital">
   <owl:equivalentClass rdf:resource="&geography;Capital"/>
</owl:Class>
```

Disjointness of Classes. To define a set of disjoint classes, the declaration `owl:disjointWith` is used. If an instance is a member of one class, it cannot simultaneously be a member of the disjoint class. Disjointness is commonly defined for cases such as "man" and "woman," where the intersection of classes is always empty, but also for weaker separations:

```
<owl:Class rdf:ID="Sea">
   <owl:disjointWith rdf:resource="#Lake"/>
</owl:Class>
```

Enumeration of Classes. OWL also provides a means to specify a class via a direct enumeration of its members. This can be done by using the `owl:oneOf` construct:

```
<owl:Class rdf:ID="RoadType">
   <owl:oneOf rdf:parseType"Collection">
      <owl:Thing rdf:about="#InterstateHighway"/>
      <owl:Thing rdf:about="#FederalHighway"/>
      <owl:Thing rdf:about="#StateHighway"/>
      ...
   </owl:oneOf>
</owl:Class>
```

Cardinality Restrictions on Properties. We can impose cardinality restrictions on properties in OWL. `owl:cardinality` permits the specification of the exact number of elements in a relation. For example, we can specify `Road` to be a class with exactly one road number.

```
<owl:Class rdf:ID="Road">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#hasRoadNumber"/>
         <owl:cardinality rdf:datatype=
               "&xsd;nonNegativeInteger">1</owl:cardinality>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Richer Characteristics of Properties. OWL also allows richer typing and characteristics of properties, e.g., transitivity, symmetry, inversion, etc. If a property P1 is, for example, tagged as the inverse of P2, then, for all x and y, it holds that P1(x,y) is true if and only if P2(y,x) is true. In this way, we can assert that a class `Region` has cities and that the cities are inversely located in this region.

```
<owl:ObjectProperty rdf:ID="hasCity">
   <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isInRegion">
   <owl:inverseOf rdf:resource="#hasCity" />
</owl:ObjectProperty>
```

These are just some examples of what OWL presents as language elements concerning classes, properties, instances, and relationships between them. A complete list of OWL's language components can be found in the *OWL Web Ontology Language Guide* [Smith et al., 2004].

The goal of OWL was to overcome the shortcomings of RDFS by defining an ontology modeling language with a high expressiveness while also providing efficient reasoning support. Since expressiveness and efficiency are tradeoffs of each other [Breitman et al., 2007], three different OWL sublanguages evolved, each focusing on different requirement aspects.

OWL Full is the complete OWL language without any limitations and complete with maximum expressiveness, but lacking any computational guarantee. All language constructs can be used in any combination as long as the result is valid RDF.

OWL Description Logic (DL) limits the expressive power of OWL Full (and increases the expressive power of OWL Light). It offers all OWL constructs with certain lim-

itations such as type separation. For example, every resource can only be a class, a property, or an individual. This means that a class cannot simultaneously be an individual. OWL DL is intended for people who want maximum expressiveness, but retain computational completeness (all conclusions can be computed) and decidability (all computations will finish in finite time) [Baader et al., 2003].

OWL Light further restricts the expressive power of OWL DL. It also offers hierarchies of classes and properties, and simple constraints enable the modeling of thesauri and simple ontologies. Limitations are imposed on how classes are related to each other. For example, the OWL DL constructors `owl:oneOf`, `owl:disjointWith`, `owl:unionOf`, `owl:complementOf`, and `owl:hasValue` are not allowed in OWL Light.

Apart from a strict type separation of classes, properties, and instances, OWL DL requires that properties are either typed as *object properties* or *datatype properties*. Object properties relate objects to other objects. Hence, occurrences of object properties are used to relate an instance of a class (e.g., `State`) to another instance of a class (e.g., `Capital`).

```
<owl:ObjectProperty rdf:ID="hasCapital">
   <rdfs:subPropertyOf rdf:resource="#has"/>
   <owl:inverseOf rdf:resource="#isCapitalOf"/>
   <rdfs:domain rdf:resource="#State"/>
   <rdfs:range rdf:resource="#Capital"/>
</owl:ObjectProperty>
```

Datatype properties, on the other hand, relate objects to datatype values which are in turn occupied by RDF literals or XML Schema types. An occurrence of a datatype property can be used to relate an instance of a class to an instance of a datatype value.

```
<owl:DatatypeProperty rdf:ID="hasLength">
   <rdfs:domain rdf:resource="#River"/>
   <rdfs:range rdf:resource=
        "http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
```

OWL is defined as a vocabulary in the manner of RDF and RDF Schema, but with a richer semantics. An ontology in OWL is a collection of RDF triples that uses this vocabulary. Ontology creators should consider which of the three sublanguages of OWL best suits their requirements.

2.4 SPARQL Query Language

What SQL is to relational databases, SPARQL is to RDF. In fact, SPARQL is a query language based on SQL that accesses information stored in ontologies and ontology-based knowledge bases. Similar to SQL, it provides `SELECT` and `WHERE` clauses. The simple query template, for example,

```
SELECT ?X, ?Y
WHERE  { ?X geo:isLakeOf ?Y . }
```

retrieves all resources and values of triples with the property `geo:isLakeOf`, or any of its sub-properties, and their classes.

The *SPARQL query language for RDF* [Prud'hommeaux and Seaborne, 2007] is used to express queries across different data sources whether or not the data is stored natively as RDF or viewed as RDF via middleware. It offers facilities for querying required and optional graph patterns along with conjunction and disjunction combinations. It also supports value testing and query constraining. The results of SPARQL queries can be result sets or RDF graphs.

SPARQL is based on matching graph patterns containing triples that, unlike RDF triples, have the option of query variables in place of all three RDF terms: the subject, predicate, and object positions. The example below shows a SPARQL query to find the mountains in Colorado from a given data graph. The query consists of two parts: the `SELECT` clause which identifies the variables to appear in the result and the `WHERE` clause that provides the basic graph pattern to match against the data graph. The `WHERE` clause consists of a single triple pattern with a single variable (`?mountain`) in object position. The query will return the names of all the mountains that are defined as mountains of Colorado in the knowledge base.

```
SELECT ?mountain
WHERE {
   <http://www.ifi.uzh.ch/ddis/ont/nli/Colorado>
   <http://www.ifi.uzh.ch/ddis/ont/nli/geo#hasMountain>
   ?mountain .
}
```

The following SPARQL example query is slightly more complex. It consists of two triple patterns to match against the data graph. The first triple contains one variable `?mountain` in object position; the same variable is in the subject position of the second triple. A second variable `?height` is in the object position of the second triple. The two

triple patterns are joined by the variable `?mountain`, which has to be bound to the same RDF term. Note that we omitted complete URIs in this query. Instead, we use the typical `PREFIX` statement to declare the namespace shortcut at the beginning of a SPARQL query. This makes the writing of complex queries simpler, and, as queries do not exhibit complete URIs, also eases readability.

```
PREFIX geo:  <http://www.ifi.uzh.ch/ddis/ont/nli/geo#>
SELECT ?mountain ?height
WHERE {
   "Colorado" <geo:hasMountain> ?mountain .
   ?mountain <geo:hasHeight> ?height .
}
```

Each solution to a SPARQL query provides one way in which the selected variables can be bound to RDF terms such that the query pattern matches the data. A result set gives all possible solutions. The abbreviated result set in table 2.1 could, for example, be found to the query above.

mountain	height
Elbert	4399
El Diente	4316
Uncompahgre	4361
Shavano	4337
Grays	4349
Torreys	4349
Belford	4327
Longs	4345
...	

Table 2.1: Possible (abbreviated) result set to the SPARQL query [`SELECT ?mountain ?height WHERE { "Colorado" geo:hasMountain ?mountain. ?mountain geo:hasHeight ?height.}`] searching for mountains in Colorado and returning their names and heights.

To restrict the solutions of graph matching, which produces a solution sequence, SPARQL offers term constraints. The `FILTER` expression restricts solutions to those which the filter expression evaluates as true. It can test RDF literals or restrict them based on arithmetic expressions. The following query is an example of arithmetic filtering.

```
PREFIX geo:  <http://www.ifi.uzh.ch/ddis/ont/nli/geo#>
SELECT ?city
WHERE {
   ?city <geo:hasPopulation> ?population .
   FILTER REGEX (?population > 1000000)
}
```

By constraining the population size variable, only those instances match the query that have a population greater than 1'000'000. In addition to numeric types, SPARQL supports "`xsd:string`," "`xsd:boolean`," "`xsd:dateTime`" types and functions on them.

Query patterns generate an unordered collection of solutions (as seen in the result set in table 2.1), with each solution being a partial function from variables to RDF terms. These solutions are then treated as a sequence with no specific order. Sequence modifiers can then be applied to create another order. The solution sequence modifiers in SPARQL are:

- `order`: put the solution in a specific order
- `distinct`: ensure that solutions in the sequence are unique
- `offset`: control where the solutions start from in the overall sequence
- `limit`: restrict the number of solutions

The SPARQL query language for RDF is also a W3C recommendation and a product of *W3C RDF Data Access Working Group* [Prud'hommeaux and Seaborne, 2007]. The current version of the SPARQL language specification and a complete grammar can be found at `http://www.w3.org/TR/rdf-sparql-query/`. To execute SPARQL queries on RDF data, several tools have been developed that we will discuss in the next section along with a selection of Semantic Web tools that we judge most relevant and established, but the selection is not intended as a complete overview.

2.5 Some Semantic Web Tools

The Semantic Web, with ontologies as its conceptual model, produced the emergence of many research areas and communities. Many applications such as ontology management, ontology engineering, ontology mapping, ontology merging, ontology-based retrieval, ontology querying, ontology learning, etc. also evolved [Staab and Studer, 2004].

In addition, a variety of tools and frameworks for performing various tasks with ontologies were and still are being developed. Some of the most prominent and widely used ones are Jena, Sesame, Pellet, and Protégé, which will be introduced in this section.

2.5.1 Jena Framework

Jena is a Java framework/toolkit for building Semantic Web applications based on W3C recommendations for RDF and OWL [Jena, 2007]. It provides a programmatic environment for RDF, RDFS, OWL, and SPARQL and includes a rule-based inference engine. It is open source, and has grown out of work with the *HP Labs Semantic Web Research* (`http://www.hpl.hp.com/semweb/`).

The Jena Framework includes

- an RDF API (application programming interface),
- an OWL API,
- the RDF/XML parser ARP which allows reading and writing of RDF in RDF/XML, N3 (standing for "Notation 3," a readable language for RDF, basically equivalent to RDF in its XML syntax, but easier to write [Berners-Lee, 2006]), and N-Triples (a line-based, plain text format for encoding RDF graphs, designed as a fixed subset of N3 [Grant and Beckett, 2004]),
- in-memory and persistent storage,
- rule-based inference for RDFS and OWL,
- and a SPARQL query engine to execute SPARQL queries.

Eyeball is a tool that is also built upon the Jena framework [Eyeball, 2007]. Eyeball is a model checker used in particular for checking RDF/OWL models for common issues and problems such as illegal URIs, missing property values, and incorrect prefix mappings. There is also a graphical user interface for Eyeball, but it is still in experimental stage.

Additionally, Jena provides the RDF publishing server *Joseki*, which provides access to RDF models by URL and query [Joseki, 2007]. It is an HTTP engine that supports the SPARQL query language and protocol. An online demo of Joseki can be found at `http://www.sparql.org/query.html`.

2.5.2 Sesame Framework

Sesame is an open source framework for storage, inferencing, and querying of RDF data [Sesame, 2007]. It is supported by *OpenRDF.org*, a community site that is the center for all Sesame-related development. Sesame has been designed with flexibility in mind. It can be deployed on top of a variety of storage systems (relational databases, in-memory, file systems, keyword indexers, etc.), and offers a large array of tools to developers for broadening the power of the RDF and RDF Schema, such as a flexible access API, which supports both local and remote access, and several query languages including SPARQL.

The Sesame project also includes *Rio*. "Rio" stands for "RDF I/O". It is a set of parsers and writers for RDF that has been designed with speed and standards-compliance as its main concerns. Currently, it supports reading and writing of RDF/XML and N3, and writing of N-Triples. Rio is part of Sesame, but can also be used as a separate tool.

Elmo is a toolkit for developing Semantic Web applications using Sesame. Elmo wraps Sesame, providing a dedicated API for a number of well-known Web ontologies, for example Dublin Core ([Dublin Core Medatata Initiative, 2007]), RSS ([RSS Advisory Board, 2007]), and FOAF ([The Friend of a Friend Project, 2007]). Additionally, Elmo offers a set of tools related to the supported ontologies, including an RDF crawler, a FOAF smusher, and a FOAF validator. Elmo's API is extensible and is expected to cover a larger set of existing Web ontologies as its development progresses.

2.5.3 Reasoners

The process of deriving new facts and associations from previously known facts is called *reasoning*. In the case of the Semantic Web, it is an inference mechanism that has classes and properties as primitives in order to derive new knowledge not explicitly specified in the ontology. Different reasoning tools are available. To name a few popular ones:

- Pellet is an open source, OWL DL reasoner in Java originally developed at the University of Maryland's *Mindswap Lab* [Sirin et al., 2007], now commercially supported by Clark & Parsia LLC [Clark & Parsia LLC, 2007].
- FaCT++ is an OWL DL reasoner implemented in C++ released under a GNU public license [Tsarkov, 2007].
- KAON2 supports all OWL Lite and a subset of OWL DL including all features of OWL DL apart from enumerated classes to keep reasoning decidable [KAON2, 2007]. It is free of charge for non-commercial academic usage. The commercial version of KAON2 is *OntoBroker OWL* available at `http://www.ontoprise.de/`.

- RacerPro is a commercial OWL reasoner that supports OWL DL almost completely [Racer Systems, 2005].

In our work, we employ the Pellet reasoner to perform inference tasks (as can be seen in Chapter 4).

2.5.4 Ontology Editors

Another group of Semantic Web tools comprises ontology editors, which help engineers to build and maintain ontologies. Examples are OilEd [Bechhofer and Ng, 2006], OntoEdit [Davies et al., 2002], and Protégé [Stanford Medical Informatics, 2007]. The most popular and most sophisticated one is Protégé—a free, open source ontology editor and knowledge base framework. The platform supports two primary methods of modeling ontologies: via the Protégé Frames or Protégé OWL editors. Ontologies can be exported into a variety of formats including RDFS and OWL. Based on Java, it is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development. Many plug-ins have been developed to complement the platform, including visualization, import/export, inference and more.

2.6 Semantic Web Challenges

The Semantic Web is thought as an extension of the current Web in which information is given well-defined meaning, thereby better enabling computers and people to work in cooperation. Ideally, computers will be able to process the information available on the Web and take over tasks that users are now doing manually. The ultimate goal and benefit is a Web more adequate for users's needs than the current one. A further advantage of the Semantic Web endeavor is the creation of truly global knowledge representations unifying proprietary conceptual models of every area and enabling the coordination of different communities by global knowledge management tools [Breitman et al., 2007, McCool, 2005].

Apart from the benefits, there are some drawbacks. If computers can "understand" the information on the Web, then people, due to the formalization and modeling of information on the basis of logic, will be unable to do so. Expressive ontologies can be fairly difficult to understand for end-users, and content-creators must often locate and inspect concepts of interest in detail in order to determine whether specific concepts are suitable for their use. Therefore, effective tools for the presentation of ontological hierarchies, for example, are a must-have. The Semantic Web supports very sophisticated constructs, but should not show this complexity to its users.

It is the logic scaffolding that makes some people actually believe that the Semantic Web will remain only a vision [McCool, 2005]. The current state of the Semantic Web and its applications are not human-friendly, and only a few applications exist, hence many challenges must be mastered in order to hide the complexity of the logic and offer user-friendly interfaces that will make the vision of the Semantic Web become reality. Our thesis is intended as one small step in this long process of realizing the Semantic Web.

3

Related Work

NLIs have undergone considerable development since the 70s, but oftentimes only with moderate success [Androutsopoulos et al., 1995, Chakrabarti, 2004, Mooney, 2004] [Thompson et al., 2005], and interest in the topic has consequently decreased since the 90s. However, the necessity for robust and applicable NLIs has become more acute in recent years as the amount of information on the Internet has grown steadily and immensely, and more and more people from an ever wider population now access data stored in a variety of formal repositories through Web browsers, PDAs, cell phones etc. As such, around 2000 researchers have once more begun to address the task of building NLIs [Cimiano, 2004, Katz et al., 2002, Lopez et al., 2005, Minock, 2005] [Popescu et al., 2003].

The major difference between the early systems and this new generation of NLIs is that the latter focus on portability and performance/robustness by applying rather shallow NLP techniques compared to the costly-to-produce systems of the past that were, in most cases, laboriously tailored to one specific domain or application. A wide range of freely available resources such as lexical knowledge bases (e.g., WordNet,[1] FrameNet [2]), ontologies (e.g., MIT Process Handbook,[3] SUMO[4]), NLP parsers (e.g., Charniak Parser,[5] Stanford Parser[6]), etc. providing linguistic as well as semantic processing support can nowadays benefit the evolution of NLIs, as the implementationally intensive resources can be used off-the-shelve.

We assign NLIs and the publications in this area to three categories: (1) NLIs to databases, (2) NLIs to Semantic Web knowledge bases, and (3) guided NLIs. We consider

[1] http://wordnet.princeton.edu/
[2] http://framenet.icsi.berkeley.edu/
[3] http://ccs.mit.edu/ph/
[4] http://www.ontologyportal.org/
[5] http://www.cs.brown.edu/~ec/
[6] http://nlp.stanford.edu/downloads/lex-parser.shtml

guided query interfaces as related work because one of our project's interfaces (i.e., Ginseng) features a guided query composition design. Furthermore, we specifically discuss projects that include a usability study emphasizing the prominent role of their functionality in our work. Detailed overviews of NLIs can be found in Androutsopoulos *et al.* 1995 [Androutsopoulos et al., 1995] and Odgen and Bernick 1997 [Odgen and Bernick, 1997].

3.1 Natural Language Interfaces to Databases

In recent years, a number of well-performing, NLIs to databases emerged [Andreasen, 2003, Dittenbach et al., 2003, Frank et al., 2007, Guarino et al., 1999] [Hallett et al., 2005, Minock, 2005, Popescu et al., 2003, Tang and Mooney, 2001].

The OntoSeek project by Guarino *et al.* created a content-based information retrieval system that retrieves information from both yellow pages and product catalogs [Guarino et al., 1999]. It presents a user interface with complete terminological flexibility that is facilitated by an ontology-driven content-matching between natural language query terms and database elements. Simple conceptual graphs are used to represent both the queries and the resource descriptions, thereby yielding an approach more flexible and expressive than applying attribute-value lists. The problem of content matching is thus advantageously reduced to ontology-driven graph matching. The project shows that the combination of linguistic ontologies and structured representation formalisms not only increases recall and precision in information retrieval tasks, but also decouples the user vocabulary from the data vocabulary.

Andreasen proposes an information retrieval approach where a domain-specific knowledge base that includes a dictionary of words and an ontology of concepts is used to transform natural language queries into aggregated hierarchical expressions [Andreasen, 2003]. More precisely, unconnected natural language query words are transformed into a compound expression by grouping the words with heuristically specified quantifiers and importance allocators. The knowledge base and a path distance measure are used to expand the query expression into a set of queries that also cover similar terms. Each document in the database is transformed into hierarchical expressions and likewise stored as an index. As such, each query can be compared to each description of the documents stored in the database. A prototype has been implemented within the OntoQuery project but no evaluation was performed.

Similarly, Minock's STEP system allows natural language access to relational databases [Minock, 2005]. STEP uses a phrasal lexicon to parse the natural language input and maps the phrasal patterns to tuple relational calculus, a variant of first-order logic that allows variables to range over entire tuples. A direct translation to SQL is then possible. Neither a general nor a domain-independent grammar is used for syntactic analysis,

thereby avoiding many difficulties with ambiguities and idiosyncrasies inherent to natural language. The lexicon is also used to generate natural language answers from the tuple relational calculus to be displayed to the user as search result. The evaluation by implementation, in which the primary goal was to collect real queries rather than to measure the system's retrieval accuracy, shows that STEP does a fairly good job of satisfying user requests. However, a full evaluation would be required to confirm this, and will be achieved in a field test involving several domains [Minock, 2007].

The disciplines of Life Sciences and Health Care have recently exhibited a phenomenal growth that has generated huge numbers of large-sized data repositories. Searching these repositories and browsing through the result sets requires a great deal of domain knowledge—a reality that makes these tasks especially challenging for occasional or non-expert users. NLIs with a high retrieval performance that support querying have, therefore, attracted considerable interest from these areas. The *Clinical e-Science Framework* (CLEF) by Hallet *et al.*, for example, provides a repository of detailed clinical histories [Hallett et al., 2005]. A relevant part of the project is the query editing interface that makes use of natural language generation techniques aimed at alleviating some of the problems faced by natural language while also providing an intuitive manner of querying. It is designed to answer questions relating to patterns in medical histories. Casual and moderate users require no prior domain knowledge, no expertise in database structures or SQL, and only little overall training. Query construction is performed by interacting with automatically generated natural language texts. The interface presents users with such a natural language text drawn from the database that corresponds with an incomplete query. The user simply selects textual elements, thereby editing the query until it is complete. Hence, each query is unambiguous, syntactically correct, and easily readable at every step. We could find no evaluation of CLEF, either in terms of a performance or a usability study.

The goal of Tang and Mooney's COCKTAIL system is learning to semantically parse natural language questions into logical queries enabling users to query a database using natural language [Tang and Mooney, 2001]. The *inductive logic programming* approach uses different strategies for the task of learning the semantic parser and investigates the construction of first-order, definitive-clause logic programs from a set of positive as well as negative examples and given background information. As such, it attempts to integrate the best aspects of existing inductive logic programming methods into a coherent, novel framework that stands at the intersection of the fields of machine learning and logic programming. In contrast, traditional parsing approaches use hand-crafted (by experts) parsers which are time-consuming to develop and suffer from problems with robustness and incompleteness. COCKTAIL applies various learning strategies and demonstrated to perform better than single learners, particularly in recall, in two domains: geography and job postings (see section 5.1.2 on page 116). Our NLIs are not learning approaches. We,

furthermore, insist on obviating a semantic analysis of the natural language questions, but rely on shallow NLP techniques and the extraction of semantic information from knowledge bases in order to translate English questions into formal, executable queries.

Among the NLIs to databases, the PRECISE project by Popescu *et al.* is the most closely related work to our approaches [Popescu et al., 2003]. It proposes an NLI to arbitrary relational databases allowing users to phrase queries in full English similar to NLP-Reduce and Querix. PRECISE uses a database augmented tokenization of a query's parse tree to generate the most likely corresponding SQL statement. Given a natural language question, a tokenizer first outputs a complete tokenization of the question. Each token is then stemmed and assigned the types of database elements that it could possibly match to (i.e., value, attribute, etc.). Additionally, the Charniak parser [Charniak, 2000] is used to detect relationships between the tokens. The matcher is the key component of PRECISE: it treats the issue of matching the natural language tokens to the database elements as a graph matching problem, similar to Guarino *et al.* [Guarino et al., 1999]. First, the database elements selected by the matcher are assembled into a SQL query. If two or more possible queries are found, PRECISE asks, in a manner similar to Querix, that the user chooses between two or more semantic interpretations of a particular token. The evaluation with the three data sets of the *Mooney Natural Language Learning Data* [Tang and Mooney, 2001] that we also used for our retrieval performance evaluation (see Chapter 5) showed that PRECISE demonstrates better recall and precision rates than COCKTAIL [Tang and Mooney, 2001], whose retrieval performance is similar to our own systems'.

3.2 Guided Natural Language Interfaces

Considering the difficulties of full natural language, it seems understandable that controlled natural language or menu-guided interfaces have repeatedly been proposed [Bechhofer et al., 1999, Cha, 1991, MKBEEM, 2002, Schwitter and Tilbrook, 2004] [Thompson et al., 2005].

The Kaleidoscope system [Cha, 1991] is an early approach that belongs to the family of so-called *menu-guided natural language interfaces*. In Kaleidoscope, SQL query creation is guided by menus. The user adds constituents to a query by selecting lexical items from a set of pop-up menus. Depending on the current state of the partial query, the system successively updates the menus. Again, no evaluation is performed manifesting the usability or the retrieval performance of the system.

Bechhofer *et al.* indicate that one major problem with NLIs is that users must know what is possible to ask in a particular domain or application and that developers, therefore, must consider how users interact with a query language when building an NLI

[Bechhofer et al., 1999]. They suggest that controlling the way in which expressions and queries are constructed enables the users to be guided in their navigation searching for appropriate query expressions. The idea was implemented in two projects: TAMBIS, a query interface for biological information sources [Baker et al., 1998], and STARCH, an interface for navigating through picture archives [Bechhofer and Goble, 1999]. Both interfaces facilitate the construction and manipulation of description logic queries. The query formulation process is driven by underlying conceptual models guiding the user towards appropriate choices and queries, although description logic is not easy for casual users. The systems isolate the user from the logic representation by a menu-based window. All query manipulations are controlled by restricting the options presented for specialization or replacement, ensuring that only reasonable queries are built—similar to Ginseng. The authors report that initial reaction to the prototypes from users have been positive, but that a formal evaluation would be conducted in the future.

One recent approach to guided query interfaces is the PENG system presented by Schwitter and Tilbrook [Schwitter and Tilbrook, 2004]. PENG is a machine-oriented controlled natural language that has a restricted grammar and lexicon. PENG is applied in combination with ECOLE, a look-ahead text editor which guides the user through the formulation of PENG sentences. The look-aheads provide syntactic hints and inform the user on how to continue the current input string. After each word form the user enters, the editor indicates what kind of syntactic structure can follow, therefore guaranteeing compliance to the rules of PENG. The user is guided by the system and does not have to learn and remember the restrictions of the controlled language, which is also the case in our Ginseng approach (see 4.3.1). The underlying framework of the PENG system consists of a complete logic-based natural language processing engine, whereas Ginseng uses a simple querying grammar that can be dynamically extended via any OWL ontology structure. Furthermore, PENG's goal is also different; it aims at providing a full natural language processing environment. Thus, knowledge must also be entered into the system using PENG's controlled language. Our interfaces, in contrast, aim at querying existing semantically annotated content.

To overcome the habitability problem, LingoLogic by Thompson *et al.* is a guided query interface technology that uses menus to specify natural language queries and commands that can be executed on relational databases [Thompson et al., 2005]. A parser constantly checks a user's entries, displays possible completions of the words or phrases to the user, and translates the entries to the target query language SQL. As such, it limits the user to performing queries that the system can "understand," as Ginseng does. The authors suggest that LingoLogic, which was designed to query databases, could also offer an extension to the Semantic Web. Ginseng represents such an extension.

3.3 Natural Language Interfaces to Semantic Web Knowledge Bases

In this section, we will discuss projects that provide natural language query interfaces to existing ontology-based knowledge bases—the area closest to our work. The increasing popularity of the Semantic Web created a number of NLIs that provide access to ontological data [Cimiano et al., 2007, Distelhorst et al., 2003, Duke et al., 2007, Frank et al., 2007, Katz et al., 2002, Lopez et al., 2005, Wang et al., 2007]. Apparently, the semantics that exists in ontologies, which can be exploited in the querying process, has reattracted the development of NLIs.

The first question-answering system with an NLI to Semantic Web data was START by Katz *et al.* [Katz et al., 2002]. The system was the noticeable beginning of the marriage between NLP and Semantic Web technology and has been in development for a period of over 10 years. Natural language annotations were considered to be intuitive and effective, and therefore perfectly suited for accelerating the pace with which the Semantic Web could be adopted. START allows annotations in natural language to describe the content of a knowledge base. To answer a user question, the system also annotates the query and compares it against the annotations derived from the knowledge base. The basic goal of the augmentation by metadata written in everyday natural language was to render the logic-based Semantic Web foundation friendlier to humans.

The GAPP project by Distelhorst and colleagues is a question-answering system developed for querying the *Foundational Model of Anatomy* (FMA) knowledge base [Distelhorst et al., 2003]. GAPP takes natural language questions as input and translates them into the structured query language StruQL, a database language designed for querying graphs. The system then returns the results of a query as an XML document. GAPP analyzes English questions and divides them into the three elements of subject, relationship, and object. Along with pattern-matching and word-combination techniques, GAPP's parser exploits the syntactic structures in order to generate appropriately structured queries. The results of the evaluation, where the generation of the correct query was considered to be a correct response, show that GAPP provides an intuitive and convenient way for anatomists to browse the FMA knowledge base. However, its query construction and its overall application are highly restricted to one semantically constrained domain. Furthermore, their model seems to be limited to a set of domain-specific user-defined pattern-matching rules. This makes the integration of new ontologies both difficult and tedious.

In the PANTO project, Wang *et al.* concentrated on the aspect of domain-independence and developed a portable NLI to Semantic Web data [Wang et al., 2007]. PANTO accepts generic natural language questions as input and executes correspondingly gener-

ated SPARQL queries on an ontology model. When a knowledge base is loaded into the system, all entities are extracted, enhanced with synonyms using WordNet, and stored in a triple lexicon. By focusing on noun phrases, the system first analyzes a user question with a statistical parser (i.e., Stanford Parser [Klein and Manning, 2002]). The parse tree is then transformed into triples and mapped to the triples in the lexicon. From the triples that match the query triples, SPARQL statements are generated and linked to one coherent SPARQL query. The approach was prototypically evaluated with the *Mooney Natural Language Learning Data*, and achieved recall performance similar to Popescu *et al.* [Popescu et al., 2003] and Tang and Mooney [Tang and Mooney, 2001]. The major advantage of PANTO is its complete portability, but the approach consequently faces difficulties with precision performance. The focus on portability also suppressed a user evaluation.

Closely related to our NLIs is PowerAqua [Lopez et al., 2005] and its predecessor AquaLog [Lopez et al., 2006a, Lopez et al., 2006b]. Both systems are ontology-driven question-answering systems that require natural language queries and return answers obtained from the semantic markup of ontology-based knowledge bases. The difference between the two systems is that AquaLog is limited to one knowledge base at a time, whereas PowerAqua handles various distributed knowledge bases simultaneously. A natural language input query is first analyzed using domain-independent linguistic tools (i.e., GATE [Cunningham et al., 2007]) and translated into a triple representation. After the transformation, a relation similarity service is invoked, which essentially tries to match the query representation to the knowledge base by using string similarity matching, WordNet, and a lexicon derived from the learning component of the system. If ambiguities in the natural language query cannot be resolved or several matches between the query representation and the triples of the knowledge base can be found, the user is asked for clarifying feedback. The learning component is based on the choices the user makes; it can, therefore, learn the jargon of a user, thereby ensuring that the system's performance improves over time for a given ontology or a particular individual. The AquaLog approach is similar to Querix and NLP-Reduce. It differs from Ginseng in that Ginseng does not apply any linguistic tools and by-passes ambiguities by only allowing queries for which the system can actually provide an answer. The overall assembly of Ginseng, NLP-Reduce, and Querix are simpler than AquaLog's, while they still manage to integrate new knowledge bases with good query processing performance (NLP-Reduce 54.35%, Querix 62.76%, Ginseng 41.53% of 1748 queries) even without modification of the systems. AquaLog, in comparison, was able to parse 48.68% of 76 queries of a new knowledge base [Lopez et al., 2005]. As far as we know, PowerAqua has not yet been entirely implemented and evaluated.

3.4 Natural Language Interfaces in Usability Studies

Most of the projects in the area of NLIs and, therefore, the evaluations of NLIs, mainly focus on retrieval performance and/or the portability dimension. As our work intends to emphasize equally the usability dimension, we will now discuss four recent NLI projects that conducted a usability study: ORAKEL, Squirrel, CHESt, and a tourism platform by [Dittenbach et al., 2003].

ORAKEL by Cimiano is a portable NLI for structured knowledge bases that is ontology-based in two ways [Cimiano et al., 2007]. First, it uses an ontology in the inference process to answer users' queries. Second, the system employs an ontology in the process of adapting the system to a domain and a specific knowledge base. This adaptation is performed by domain experts and has been evaluated in a user study. It was shown that people without any NLI expertise could adapt ORAKEL by generating a domain-specific lexicon in an iterative process. The controlled study involved 26 users from both academic and industrial institutions. Results were reported in terms of recall and precision—showing that the iterative methodology to lexicon customization was indeed successful. A second experiment was performed to determine the linguistic coverage of 454 questions asked by end-users. They reported an excellent coverage of 93%, but did not investigate the usefulness from the end-users' point of view.

The Squirrel system presented by Duke *et al.* is a search and browse interface to semantically annotated data [Duke et al., 2007]. It allows combined search facilities consisting of keyword-based and semantic search in order to balance between the convenience for end-users and the power of semantic search. Users can enter free text terms, see immediate results, and follow up with a refinement of their query by selecting from a set of matching entities that are associated with the result set returned by the system on the basis of an ontology. Squirrel has been evaluated in three steps: (1) in a heuristic evaluation, in which usability experts judged the interface according to a list of usability heuristics, (2) in a walk-through evaluation, where users were asked to complete a number of tasks, while their actions were recorded, and (3) in a set of field tests that gave users information-seeking tasks and collected feedback. Promising results obtained from the field tests with 20 users were reported: Squirrel achieved an average perceived information quality score of 4.47 on a 7-point scale. It was rated positively regarding its properties, but skeptically in terms of performance and speed. Regrettably, the authors provided neither a detailed description of the evaluations nor explicit results.

The core of the work by Reichert and colleagues lies in a usability study, making it most related to the our work [Reichert et al., 2005]. They investigated how students assess the possibility of querying a multimedia knowledge base by entering full questions instead of just keywords. For this purpose, two versions of the e-learning question-

answering tool CHESt [Linckels and Meinel, 2005] were implemented. The first version offers a keyword-based search; the second version allows a semantic search with full sentences as query input. They conducted three task-oriented experiment sessions with 18, 18, and 14 students and benchmarked the two versions of CHESt against each other. The outcome of the three sessions was that the students generally preferred the keyword-based search to the full questions search (76% on average). This was found to be independent of the appropriateness of the results. The students reported that they would use the option of complete questions if this yielded better results. Nonetheless, the authors concluded that the intellectual task of thinking and formulating full sentence queries must not necessarily be considered burdensome compared to simply entering loose keywords. We will confirm this conclusion in our usability study (see Chapter 6), which presents a wider choice of query languages, and we will draw even more detailed conclusions.

The last approach we want to mention is concerned with the general usefulness of NLIs. Dittenbach *et al.* developed a natural language query interface that was designed to exploit the intuitiveness of natural language for a Web-based tourism platform [Dittenbach et al., 2003]. The system identifies the relevant parts of a natural language query by using an ontology that describes the domain as well as the linguistic relationships between the concepts of the domain. The ontology also contains parametrized SQL fragments that are used to build the SQL statements representing the natural language query. A lightweight grammar is used to analyze the structure of a question and to combine the SQL statements in order to obtain one coherent SQL query that can be executed over the database. The interface was integrated into the *Tiscover*[7] platform , and was online for ten days. The goal was to collect a broad range of questions and to discover what users really wanted in an unsupervised field test. 1425 unique queries were collected in two languages—German and English. In 57.05% of the queries, users formulated grammatically correct and complete queries, whereas only 21.69% used the interface like a keyword-based search engine. The remaining queries (21.26%) were question fragments such as "double room for two nights in Vienna." It was reported that the users accepted the NLI and were willing to type more than just keywords to search for information; some queries even consisted of more than one sentence. The authors assume that users are more specific formulating a query in natural language than with keywords, a conclusion we can confirm on the basis of our controlled usability experiment. In general, the study shows that the complexity of natural language questions is relatively low, i.e., the number of concepts that are combined in queries is low (the average number of relevant concepts occurring in the queries was 3.41), and the questions formulated on the basis of combining concepts are of simple syntactical manner. Hence, the complexity of the sentences expressing the user's information need is tractable with shallow language

[7] http://www.tiscover.at/

processing techniques. Motivated by these findings, we more than ever tried to keep our NLIs simple in design and avoid complex configurations by extracting the information needed to process natural language queries from the underlying knowledge bases.

3.5 Summary of Related Work

In summary, we can say that existing NLIs to databases that allow full natural language input are in most cases restricted to the domain of the queried database. Furthermore, they require computationally costly and conceptually complex algorithms. The adaptation to new databases can only be accomplished by lengthy manual reconfiguration. Outside controlled domains, most of the NLP systems and undertakings are very ambitious goals. NLI research has, not surprisingly, gradually moved toward building robust tools for simpler tasks [Chakrabarti, 2004]. Guided query systems aim at a variety of objectives. Those querying databases consequently suffer from the same problems as the NLIs to databases, but also relieve the user from having to learn a formal query language or to experience the effect of the habitability problem that can occur when full natural language is admitted as the query language. The application of NLIs to Semantic Web knowledge bases exploits the semantic information that is enclosed in ontologies in query analysis and translation. Similar to our interfaces, most current approaches make use of off-the-shelf tools and build on shallow techniques from an NLP perspective in order to achieve robustness, portability, and high-quality retrieval performance. Only a minority of the above approaches include a full usability evaluation, and, if they do, their subjects are mostly students who do not represent the general public. The generalization of their conclusions is, therefore, uncertain; in other words the external validity of the conclusions is not satisfied [Bortz and Döring, 2002].[8]

Approaches in the field of NLIs to the Semantic Web demonstrate that such interfaces can successfully tackle the retrieval performance and transportability dimension. As such, they complement our hypothesis that NLIs to the Semantic Web can be built by extracting the necessary information to process natural language queries from the underlying ontology-based knowledge bases, since these knowledge bases offer a rich source of semantically annotated information. Consequently, our approaches exploit even further the reduction of the complexity and technical setup of these search systems. The competitive edge of our NLIs is thus their captivating simplicity.

Besides achieving conceptual and technical objectives, our work aims at thoroughly investigating the usability of our proposed approaches by focusing on the usability di-

[8] *External validity* is the degree to which the conclusions from a study can be generalized from a specific sample to a larger group. As such, a study with high external validity allows its findings to generalize to the population at large.

mension. We even attach a similar importance to the habitability hypothesis and its evaluation by asking real-world users what query languages they actually find useful. Some of the approaches investigating usability confirm our findings that NLIs are useful from a casual end-user's point of view, and particularly useful for inhomogeneous user groups. However, only very few recent usability studies concerning NLIs to ontology-based data exist, and studies evaluating controlled query languages are even rarer if not inexistent in the field of the Semantic Web. Hence, more work is needed on NLIs to Semantic Web data and further comprehensive usability studies in order to investigate the end-users' perspective.

Motivated by this literature review, we believe that NLIs to Semantic Web knowledge bases have the potential to be very useful. As ontologies contain a great amount of semantic information by defining classes and relationships (and their types), their contents offer a rich platform for use in the querying process. Particularly the relationships in ontologies carry structural meaning, which enables reasoning, whereas the semantic information of attributes and values in databases is clearly limited. Nevertheless, modern NLI projects can benefit from the experiences, techniques, and tools from database query interfaces [Badia, 2007].

4

Four Different Query Interfaces to the Semantic Web

To support and evaluate our propositions that (1) NLIs can perform well in retrieval tasks and be domain-independent without being unnecessarily complex (summarized as adaptivity hypothesis) and that (2) controlled query languages lying somewhere in the middle of the Formality Continuum are well-suited for casual end-users in information seeking tasks (outlined as habitability hypothesis), we developed and implemented four different query interfaces to Semantic Web data (i.e., OWL-based knowledge bases): *NLP-Reduce*, *Querix*, *Ginseng*, and *Semantic Crystal*. Each of the four interfaces requires a different query language to regulate its freedom, naturalness, structuredness, and formality: ranging from keywords to complete English sentences, from menu-based options to a graphically displayed query language. As such, the four systems cover different positions along the Formality Continuum as shown in Figure 4.1.

The overarching goal when developing the systems was to bridge the gap between the Semantic Web and the average user, who is mostly unwilling or unable to command the formal logic that provides the stable scaffolding of the Semantic Web. Since querying is a major interaction mode with ontology-based data for end-users, bridging it is, therefore, central for the success of the Semantic Web. To that end, we developed three interfaces with natural language as the query language and one interface that employs a formal, but graphically displayed, clickable query language.

Consequently, one interface, NLP-Reduce, allows almost any natural language question input, therefore lying at the natural language end of the Formality Continuum, whereas the second interface, Querix, narrows the input to a controlled set of natural language questions with regard to sentence beginnings. Ginseng, the third interface, even controls a user's input via a fixed vocabulary and predefined sentence structures. Moreover, it constantly checks a user's entries in order to prevent entries not compliant with the system's query language in advance. The fourth interface, Semantic Crystal,

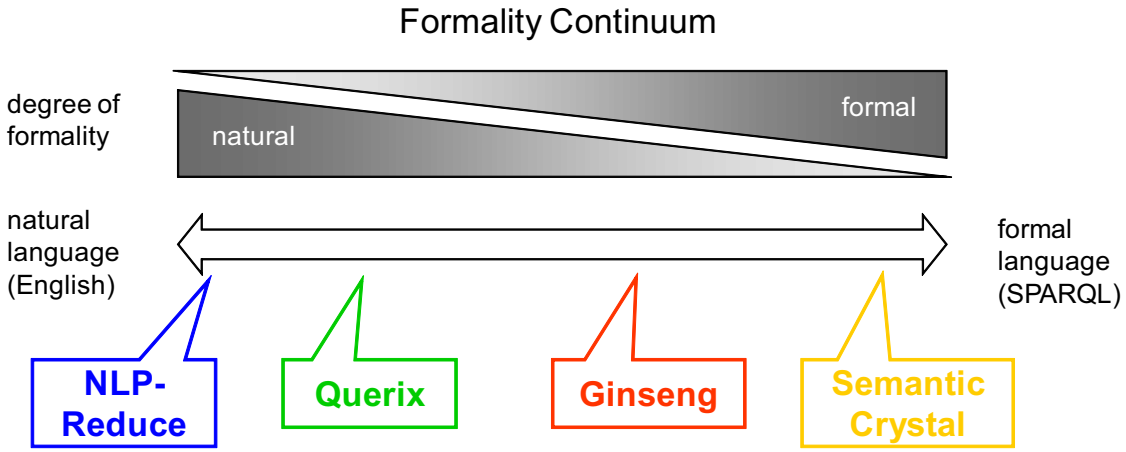


Figure 4.1: The four query interfaces NLP-Reduce, Querix, Ginseng, and Semantic Crystal to query Semantic Web data exhibit different query languages with regard to their degree of freedom, naturalness, structuredness, and formality, therefore providing different positions distributed along the Formality Continuum.

possesses the most formal query language. However, it employs ideas of visualization techniques to simplify the usage of a pure formal query language approach by graphically displaying the query language to the user. As such, Semantic Crystal is not an NLI; it veers towards the opposite, hence formal, end of the continuum. With these query interfaces at hand, we will be able to gain some insight into the habitability problem and investigate our hypothesis to where on the Formality Continuum the most approved or most despised query interfaces for casual end-users, with particular regard to their respective query languages, lie.

To support the adaptivity barrier hypothesis, all four interfaces were made as domain-independent and adaptive to arbitrary OWL knowledge bases as possible by extracting the necessary underlying frameworks automatically from those knowledge bases. As such, the technical setup of our systems is similar. They basically preprocess and analyze a user question, match the question to the content of a knowledge base that has been prepared for said matching, and translate the matches into statements of a formal query language (i.e., SPARQL) in order to execute the query. However, the interfaces' query languages determine the technical setup of the four systems. The more flexible and less controlled a query language is, the more complex a system's question analyzing component needs to be in order to compensate for the freedom of the query language by embedding sophisticated NLP tools such that good retrieval results can be achieved. The more formal and controlling a system's query language is, the simpler its technical groundwork in general. However, a query language that is supposed to guide a user, thereby appropriating a major component of both query composition and query processing load, demands rather complex and challenging knowledge base preprocessing

techniques (which is the case with Ginseng). Nevertheless, in order to fulfil our hypothesis, we sought to avoid complex and tedious full NLP scaffolds on one hand as well as a formal query interface design that shifts all "intellectual" query construction work to the user on the other hand. Furthermore, the overall design of all interfaces should conform to the interaction design paradigm [Preece et al., 2002], thus enhancing and supporting the way users communicate with Semantic Web repositories.

In the following, we will describe each of the four systems along the Formality Continuum beginning with the interface that has the least restrictive and most natural query language (NLP-Reduce), then continuing with the systems that feature more controlled query languages (Querix and Ginseng), and closing with the system requiring a formal, graphical query language (Semantic Crystal). For each system, we will give an introductory sketch of the interface, show how a user experiences query formulation, present the system architecture and a technical overview, and follow up with more details on the interface's functionality.

4.1 NLP-Reduce

NLP-Reduce is a "naïve" and completely portable NLI for querying Semantic Web knowledge bases [Fischer, 2006, Kaufmann et al., 2007]. It is called *naïve* because the approach is simple and processes natural language queries as bags of words—it employs only a reduced set of NLP techniques, such as stemming and synonym expansion.

The interface uses the least restrictive and most natural query language of our four interfaces, and therefore lies furthest towards the natural query languages' end of the Formality Continuum. Users can enter keywords (e.g., "size cities Illinois"), sentence fragments (e.g., "size of the cities in Illinois"), or full English sentences (e.g.,"How big are the cities in Illinois?"). The interface is, therefore, highly robust for deficient or ungrammatical input.

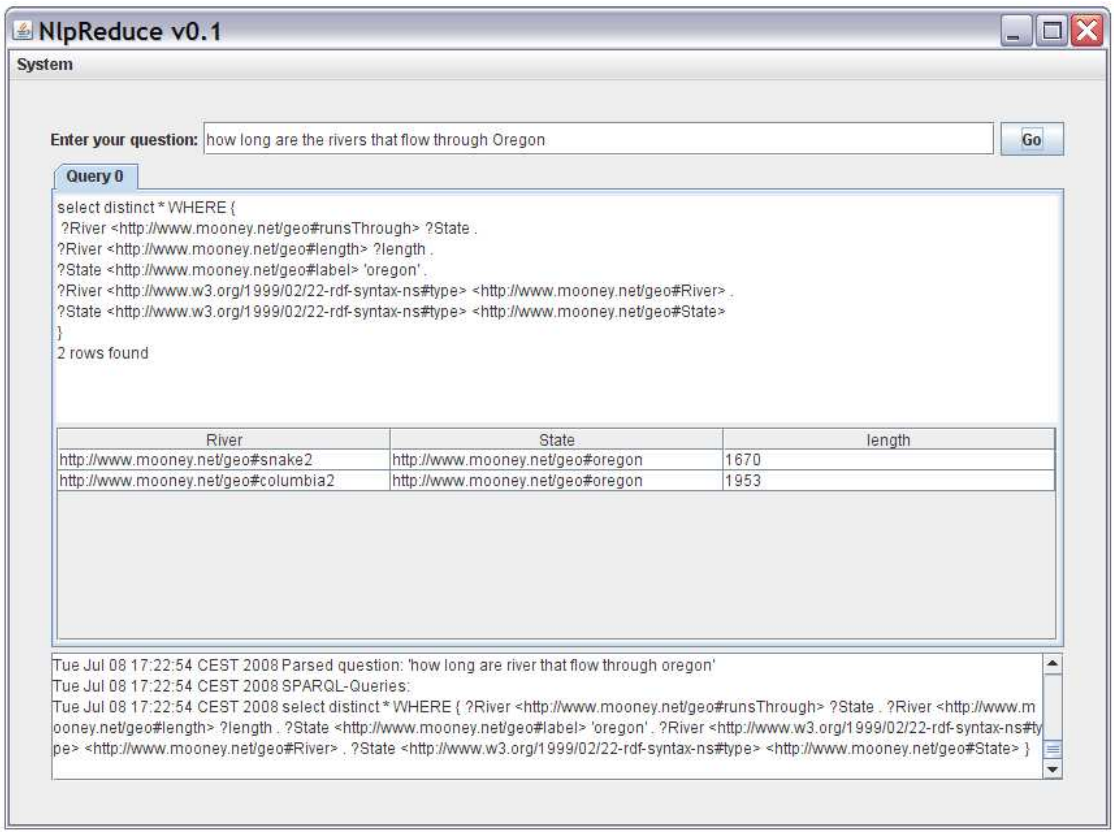


Figure 4.2: The NLP-Reduce user interface after executing the query: "how long are the rivers that flow through Oregon".

A query entered by a user is first reduced by removing stopwords as well as punctuation marks and stemming the rest of the words (accentuating the name *NLP-Reduce*

even more). The system then tries to match the remaining query words to the synonym-enhanced triple store that is generated from an OWL knowledge base when it is loaded into NLP-Reduce. Next, it identifies triple structures in the matches, and finally joins and translates the identified triples into SPARQL statements. To execute the SPARQL query, NLP-Reduce uses the Jena framework [Jena, 2007] and the Pellet Reasoner for reasoning [Clark & Parsia LLC, 2007]. After executing the query, the SPARQL statements, the results (including the full URIs), and some execution statistics are displayed to the user (as shown in Fig. 4.2).

When generating the triple store from a knowledge base, NLP-Reduce also obtains synonyms from WordNet [Miller et al., 1993], thus providing users with a larger deployable vocabulary. This leads to better usability and eases the interface's limitation of being dependent on the quality and choice of the vocabulary used in knowledge bases. This weakness, however, is also the interface's major strength, as it does not need any adaption for knowledge bases as long as the knowledge bases are specified in OWL, therefore being completely portable. From an end-user's point of view, the major advantage of the system is that the query language is extremely flexible and robust to ungrammatical or deficient input.

To avoid a tedious, lengthy configuration phase and complicated algorithms, we intended to reduce the system's complexity to a minimum as much as possible. As such, NLP-Reduce forgoes a full linguistic analysis of queries in favor of surface level preprocessing, which is paired with ontology-augmented matching elements. Compared to a full NLP engine, it employs just two very basic NLP techniques: stemming and synonym expansion. Certainly, NLP-Reduce does not claim to be "intelligent" by interpreting and understanding the input queries; it only tries to link the words of a query to the expressions and their synonyms used in a knowledge base. We still believe, and our opinion was confirmed by the retrieval performance evaluation (see Chapter 5), that the simple, domain-independent approach provides a computationally cheap chance to offer the Semantic Web's capabilities to the general public.

The following sections will first introduce how users interact with NLP-Reduce, next present the technical design of NLP-Reduce, in particular the building of the lexicon that provides the basis for matching a query to the contents of a knowledge base, and the query generator that accomplishes the matching of a query to a synonym-enhanced knowledge base.

4.1.1 Querying in NLP-Reduce—The User Experience

The design of NLP-Reduce's graphical user interface is simple in order to be suitable for casual end-users. Nevertheless, we decided to show the generated SPARQL queries and

display full URIs in the result set for the case that some users are familiar with Semantic Web basics and, therefore, interested in additional information. This was a poor decision, as the additional information confused more users than were able to appreciate it. Fortunately, the appearance of the prototype implementation could be easily changed in order to eliminate the current deficiency.

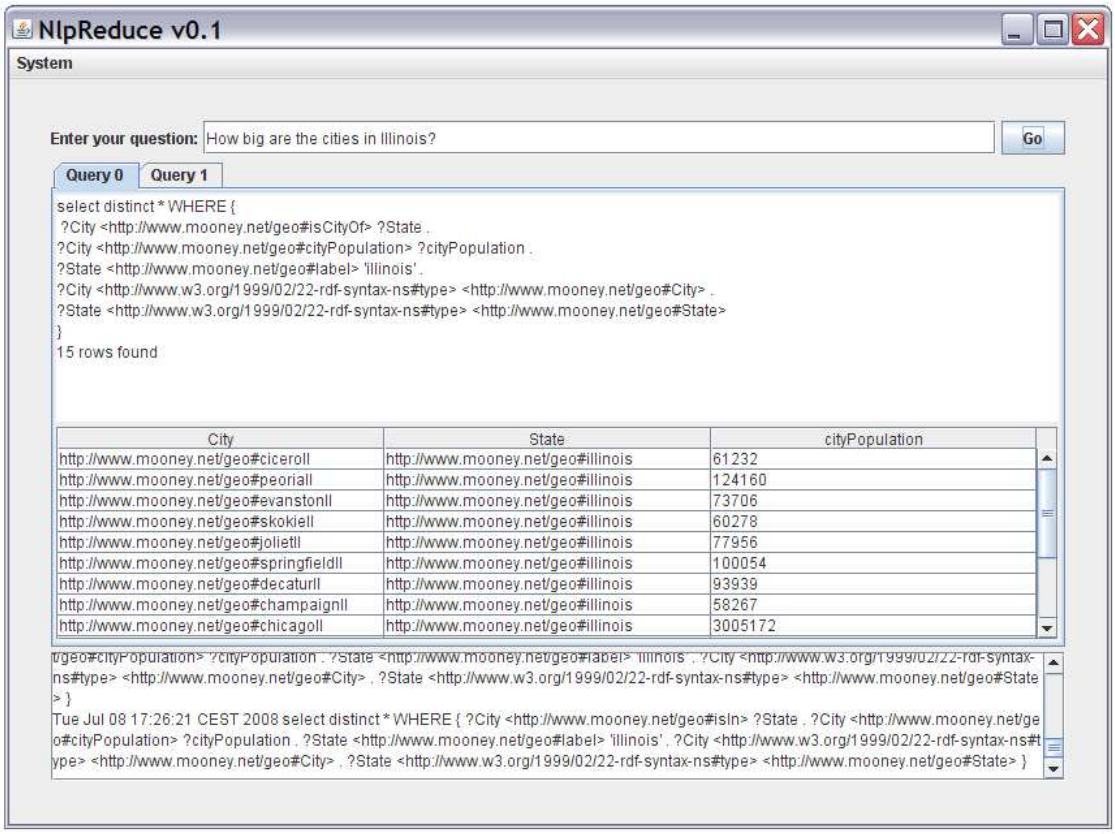


Figure 4.3: The NLP-Reduce user interface after executing the query "How big are the cities in Illinois?" showing the generated SPARQL query, the answer set to the query with full URIs in the form of a table, and some execution statistics on the bottom of the user interface.

Basically, the interface requires no training. A user simply types a query into the free text entry field at the top of the interface (cf. Figure 4.3). Punctuation marks are not required and can be completely disregarded. Case sensitivity is not required either. By clicking on the button that is labeled "GO" or by pressing the "Return/Enter" key on the keyboard, the search process is started. If different SPARQL queries can be generated returning different answers, then those SPARQL queries and their corresponding answer sets can be viewed by opening the tabs that are displayed below the query entry field and labeled by "Query" and a number. Moreover, the number of rows (e.g., "15 rows found")

actually indicating the number of results that could be found to a query is presented for every SPARQL query. As such, every question beginning with "How many ..." or "What is the number of ..." is immediately provided with the appropriate answer.

Below the SPARQL query field is a table that contains the answers. For each variable occurring in the SPARQL query, a column headed by the ID (the name) of the resource that has been bound to the variable is depicted in the middle of the user interface. The field on the bottom of the interface shows execution statistics, such as the full question entered by the user, the question after removing stopwords and stemming (i.e., the parsed question), the generated SPARQL queries, and date/time information. In case the engine does not generate a SPARQL query, "No hits found!" is shown in the execution statistics field.

NLP-Reduce's major advantage for users is the robustness and freedom of its query language. Users have the opportunity to enter arbitrary keywords, sentence fragments, and full sentences—all of which need not be grammatically correct. Moreover, they can choose whether they wish to use punctuation marks or completely discard them. But, obviously, the freedom increases the habitability problem. NLP-Reduce does not provide any query formulation guidance, thereby potentially causing users to repeatedly select words that return an answer via a backtracking behavior, and therefore supporting our habitability hypothesis. We tried to overcome the problem by embedding WordNet, facilitating the use of synonyms and, therefore, also extending the querying vocabulary, however the interface, due to its flexible natural query language, is inherently affected by the habitability problem. The benefit, on the other hand, is a fully portable NLI.

4.1.2 Technical Overview

NLP-Reduce belongs to the so-called *pattern-matching systems*. Such systems check an input for the presence of constituents of a given pattern. The main advantage is their simplicity. A question entered into a pattern-matching query interface, for example, does not require full syntactic analysis. Hence, even ungrammatical questions can be processed. However, in order to match an input to some data repository, the system needs to "know" and preprocess the data in the repository. NLP-Reduce's technical setup hinges on pattern-matching techniques in order to match a question to an OWL knowledge base.

On the other hand, there are *rule-based* or *grammar-based systems*. Most rule-based systems analyze the syntactic structure of an input according to a grammar that specifies the possibilities. These systems are usually rather limited with regard to the set of syntactic structures and also domain-dependent, though grammars can certainly be extended in general. Our goal to develop an interface featuring a maximally flexible query language and a minimally complex as well as minimally laborious configuration disqualified the

grammar-based approach. The simplicity of a pattern-matching strategy was found to be an ideal baseline for NLP-Reduce.

The architecture of NLP-Reduce consists of the following components: a user interface, a two-part lexicon, an input query processor, a SPARQL query generator, the Jena framework as an ontology access layer including its own SPARQL query execution engine, and the Pellet reasoner (see Figure 4.4).

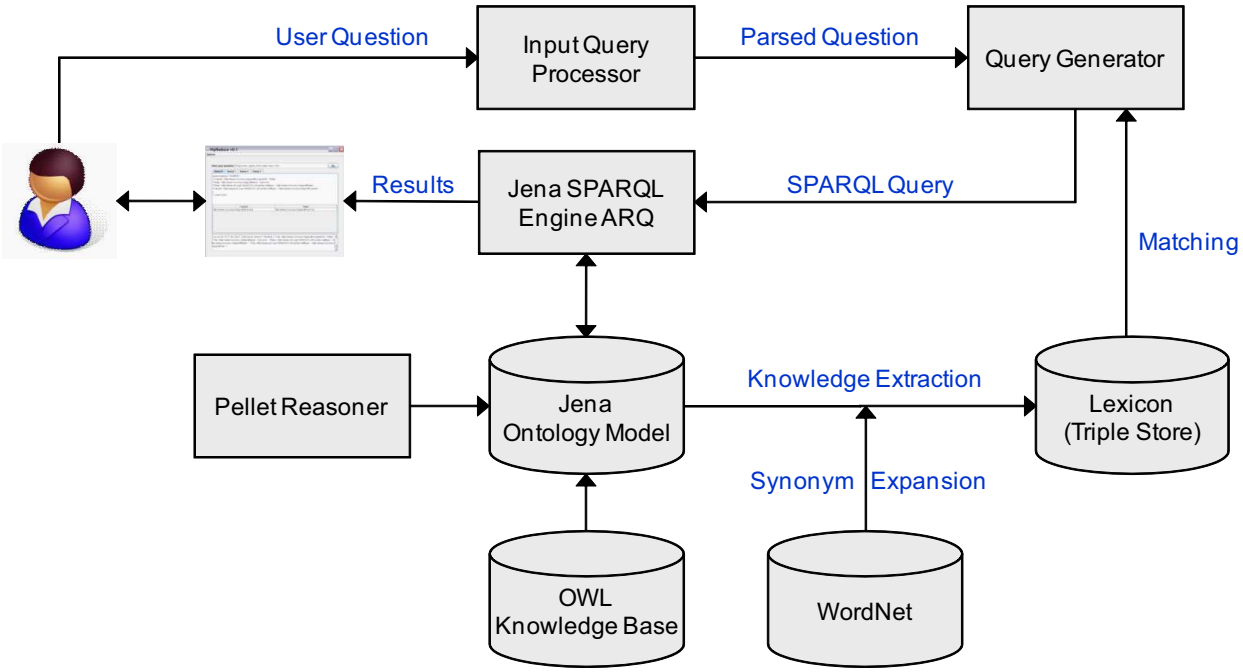


Figure 4.4: The NLP-Reduce architecture exhibiting a question parser and a query generator that matches the parsed question to the lexicon. The lexicon (i.e., an indexed triple store) is extracted from an OWL knowledge base and extended with synonyms from WordNet. The Jena framework is used as ontology-access layer and SPARQL execution engine.

The *user interface* allows the user to enter full natural language queries, sentence fragments, or even just keywords as described in the preceding section. After executing a query, it displays the generated SPARQL query, the results to the query, and the execution statistics to the user.

When starting NLP-Reduce, an OWL knowledge base located in a specified file path is first uploaded into the Jena ontology model, which allows the easy export and transformation of a knowledge base in triple format. The *lexicon* is then automatically built by extracting all explicit and inferred subject–property–object triples that exist in the Jena ontology model, including domain and range information, with SPARQL queries. In order to infer inexplicitly stated triples from the knowledge base and to enclose them

in the lexicon, the Pellet reasoner is applied [Clark & Parsia LLC, 2007]. We decided to use Pellet because it supports OWL DL and can be integrated straightforwardly into the Jena framework. For each triple extracted from the model, the synonyms of the IDs in the ontology model are obtained from WordNet [Miller et al., 1993], thus providing an extended vocabulary that can be deployed by users when querying. To improve the system's retrieval performance with respect to recall (i.e., finding all the answers in the knowledge base that are relevant to a query), each word in the lexicon is stemmed using the Porter Stemmer [Porter, 1980]. The Porter Stemmer was first introduced in 1979[1] and revised in 2006[2] as an algorithm for the process of removing common morphological and inflexional endings from English words by means of suffix stripping.

The *input query processor* of NLP-Reduce first reduces a query by removing stop words and punctuation marks. It then passes the stemmed words to the query generator, completely ignoring the syntactic structure of the question. The *query generator* essentially attempts to match the parsed and reduced question words to the synonym-enhanced triples stored in the lexicon, and generates SPARQL statements for those matches. Furthermore, it concatenates the SPARQL statements into one coherent SPARQL query based on domain and range information that is also stored in the lexicon. To execute the generated SPARQL query, NLP-Reduce uses the Jena framework as ontology access layer and its SPARQL query engine ARQ. As NLP-Reduce was implemented in Java, the application of the Jena framework for Java was convenient and appropriate.

4.1.3 The Lexicon Extraction

When starting NLP-Reduce, and thereby also loading an OWL knowledge base into a Jena Model, a synonym-enhanced lexicon is simultaneously generated. This lexicon provides the basis for matching a question with the contents of the knowledge base. All explicitly stated triples as well as those triple statements that can be inferred from the Jena Ontology Model using the Pellet reasoner are then passed to WordNet, which supplies synonyms for the words occurring in the triples. Finally, the synonym-enhanced triples and their domain/range information are stored as a triple store in what we call the *lexicon*.

In fact, the lexicon is created in two steps, and therefore consists of two parts. In the first step, all triples containing object or datatype properties are extracted from the Jena model. For each noun or verb used in a triple, the corresponding synonyms are obtained from WordNet, thus generating additional triples to be used in the matching process. All extracted and generated triples, including their domain and range information, are

[1] `http://tartarus.org/~martin/PorterStemmer/index-old.html`
[2] `http://tartarus.org/~martin/PorterStemmer/`

stored in the lexicon's part 1. A triple such as

```
geo:Lake geo:isLakeOf geo:State ,
```

being part of a class definition, would be stored in lexicon part 1 as a triple with the worded information that the object property `geo:isLakeOf` has a domain that is a class of type `geo:lake` and a range that is a class of type `geo:state`. The synonym-derived triples are stored similarly. The words occurring in the lexicon's triples are also stemmed with the Porter Stemmer and used as a word index for lexicon part 1 (similar to the index of Andreasen's approach [Andreasen, 2003], see section 3.1).

The second part of the lexicon is built of the literals that occur in the knowledge base. Each literal, its value, and its corresponding class name are extracted from the Jena Model and stored in lexicon part 2. As such, the instance

```
<Lake rdf:ID="michigan">
   <label>michigan</label>
   <lakeArea rdf:datatype="xml:float">58016</lakeArea>
</Lake>
```

generates two entries to be stored in lexicon part 2. The first entry contains the information that an instance of class type `geo:Lake` has a literal of type `geo:label`, and that the literal's value is "`michigan.`" The second entry refers to the datatype property `geo:lakeArea`, which contains the information that an instance of class type `geo:Lake` has a literal of property type `geo:lakeArea` whose value is "`58016.`" Obviously, it does not make sense to stem values, as they comprise proper names or numbers, hence the values in lexicon part 2 are used as a literal index without being stemmed.

The partition into two lexicon parts is not only beneficial for processing speed, but also relevant because different SPARQL statements are required to retrieve adequate answers for queries searching for resources and queries searching for literal values. A SPARQL statement searching for the object property `geo:isLakeOf`, for example, appears as

```
?lake geo:isLakeOf ?state .,
```

whereas a SPARQL statement searching for a specific value of a literal has to be specified as

```
?instance geo:label "michigan" .
```

We will come back to this distinction in the next section, where we explain the functionality of the query generator and the generation of SPARQL queries from retrieved lexicon triples.

One prerequisite for NLP-Reduce to be useful is that ontologies should be complete, and reasonable names (IDs) were consequently chosen for the resources. The more meaningful the IDs are, the wider NLP-Reduce's vocabulary is for querying knowledge bases and, consequently, the more appropriate the retrieval results are. We tried to overcome the ontology dependency in three ways: first, by integrating WordNet in order to semantically/conceptually augment a knowledge base's vocabulary; second, by having NLP-Reduce make use of OWL-typed properties such as transitive, symmetric, and inverse properties in order to allow the Pellet reasoner to deduce additional triple statements that subsequently become part of the lexicon; and third, by allowing NLP-Reduce to use synonyms that are included directly into the ontology (similar to the tag "`ginseng:phrase`" in Ginseng, cf. Section 4.3.4). Consider the object property `geo:isMountainOf` defined as the following:

```
<owl:ObjectProperty rdf:ID="isMountainOf">
   <rdfs:subPropertyOf rdf:resource="#isIn"/>
   <rdfs:domain rdf:resource="#Mountain"/>
   <rdfs:range rdf:resource="#State"/>
   <reduce:phrase rdf:value="is in"/>
   <reduce:phrase rdf:value="is located in"/>
   <reduce:phrase rdf:value="in"/>
   <reduce:phrase rdf:value="lies in"/>
</owl:ObjectProperty>
```

NLP-Reduce's design allows for any synonyms of IDs such as "`isMountainOf`" that are defined in an ontology model to be included in the ontology by annotating the synonym expressions with the tag "`phrase`" from the "`reduce`" namespace. As a consequence, all tagged synonyms become part of the lexicon when loading the knowledge base and can, therefore, be used when matching query words to the triples stored in the lexicon. The word sequences "is in," "is located in," "in," and "lies in," for example, are recognized as being synonyms of "is mountain of" and, therefore, matched to questions such as "What mountains are in Alaska?". As such, NLP-Reduce facilitates knowledge base-specific adaptation if the need for this should arise while also featuring an overall domain-independent technical design. Note that the mechanism not only tackles the adaptivity barrier, but also alleviates the habitability problem caused by the freedom of NLP-Reduce's query language by increasing the vocabulary of both the query language as well as the knowledge base.

4.1.4 The Query Generator

The query generator is the core component of NLP-Reduce. It accommodates the matching between the reduced question words and the synonym-enhanced triples stored in the lexicon. Moreover, it generates corresponding SPARQL queries for the matches. We will explain how the query generator works by working through the example question "How big are the cities of Illinois?". Figure 4.5 shows each step performed by the query generator when matching a question to the two-part lexicon.

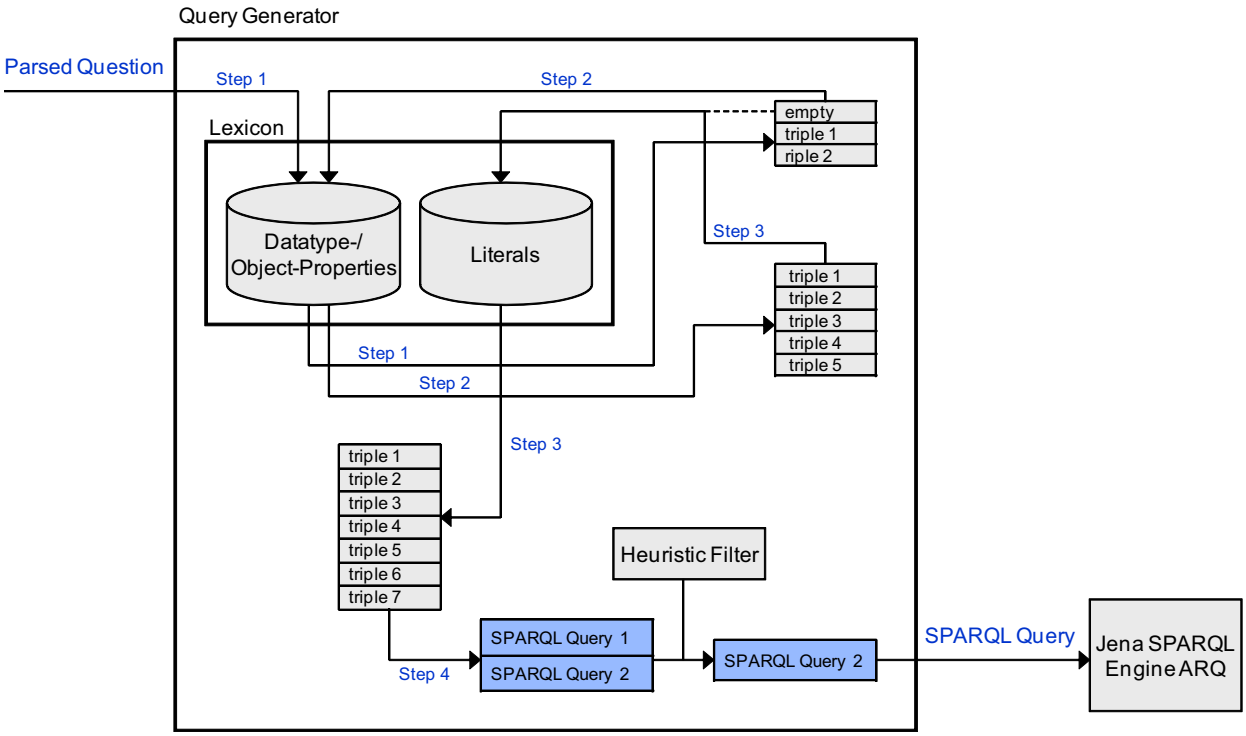


Figure 4.5: The NLP-Reduce query generator matches a reduced and parsed question to the triples stored in the two-part lexicon by looping through the lexicon's triples in three steps. It then joins the retrieved triples in compliance with the domain and range restrictions of the triples' properties. Finally, it generates corresponding SPARQL query translations for the join of the triples.

For the question "How big are the cities of Illinois?", the following four steps are performed by the query generator (to keep things clear and traceable, we only instance triples that originated from the resources stated in the knowledge base not including those triples which were generated as part of the synonym expansion by WordNet):

Step 1. When receiving the parsed and reduced question "big be city of illinois" from the input query processor, the query generator first searches for matching triples

in the word index of lexicon part 1. It retrieves all triples in which at least one of the question words occurs within an object property. For the example question, the triples

```
domain=[geo:City] geo:isCityOf range=[geo:State],
domain=[geo:Capital] geo:isCapitalOf range=[geo:State],
domain=[geo:HighestPoint] geo:isHighestPointOf
                range=[geo:State],
domain=[owl:Thing] geo:isIn range=[owl:Thing]
```

are returned, as their properties either contain the verb "is"— which is a word form of "be" like the question word form "are"— or the word "city" or the word "of." The query generator ranks the found triples according to a rating system that favors triples that cover more words as well as triples whose word stems show a better agreement with the query words over others. For example, triples containing `geo:isCityOf` receive a higher score than `geo:isCapitalOf`, `geo:isHighestPointOf` and `geo:isIn` due to their inclusion of the two question words "city" and "of."

Not every question necessarily leads to a triple comprising a datatype or an object property. A user asking a query such as "Give me all cities!" expects an answer that simply lists the names of all cities in the knowledge base. It is irrelevant where the cities are or what their populations are. In order to answer these question types, NLP-Reduce allows for an empty solution after step 1. If an empty solution is found, step 2 is skipped, and it proceeds directly to step 3 as indicated by the dashed line in Figure 4.5.

Step 2. The generator then searches for properties in lexicon part 1 that can be joined with the triples found in step 1 by the remaining question words. In our example, it searches for the query words "big" and "Illinois" in order to combine them with the triple set identified by step 1 taking domain and range restrictions into consideration. Since "big" can be related to "size," which in turn is related to "population" and "area" through their synonyms "population size" and "size," the generator finds the triples

```
domain=[geo:City] geo:cityPopulation range=[xml:float],
domain=[geo:Lake] geo:lakeArea range=[xml:float], and
domain=[geo:State] geo:stateArea range=[xml:float].
```

Next, the generator combines the triples found in step 2 with the triples from step 1. The only possible combination that is accepted by the domain and range restrictions of the triples is between `geo:isCityOf` and `geo:cityPopulation`, where the joining element is the class `geo:City`. The combined triples are, at this stage, thus:

```
domain=[geo:City] geo:isCityOf range=[geo:State],
domain=[geo:City] geo:cityPopulation range=[xml:float].
```

The matching and combining of triples deriving from lexicon part 1 is now complete.

Step 3. Since there is still a query word left in the parsed question, the query generator proceeds to the lexicon's part 2, where the triples containing literals and values from the knowledge base are stored. The generator searches the literal index for all datatype property values that match the remaining word of the query. For the last query word in our example, "Illinois," the triple

```
domain=[geo:State] geo:label range=["illinois"]
```

is retrieved. The rating system again ranks best matches. In the case of our example, only one appropriate triple could be found; the ranking is, therefore, redundant. The triples found in part 2 of the lexicon must be combined with the previously identified and joined triples. The combination must again conform to domain and range information similar to the combination process in step 2. The class `geo:State` with the label `"illinois"` can be combined with the property

```
domain=[geo:City] geo:isCityOf range=[geo:State]
```

from step 1, which has the range `geo:State`, therefore providing a joining element, as `geo:State` also occurs as domain of the "Illinois" triple. The result of the successful combination of the retrieved and incrementally joined triples from steps 1 to 3 is:

```
domain=[geo:City] geo:isCityOf range=[geo:State],
domain=[geo:City] geo:cityPopulation range=[xml:float],
```

```
domain=[geo:State] geo:label range=["illinois"].
```

Step 4. As there are no more query words left, the last step performed by the query generator is the composition of the corresponding SPARQL query statements with the appropriate variable bindings for the joining of the retrieved triples that achieved the highest score in steps 1 to 3. Additionally, it removes duplicates and passes the SPARQL query to the ontology access layer.

As NLP-Reduce tries to neither syntactically nor semantically interpret a question, each SPARQL query starts from the following query template:

```
SELECT DISTINCT *
WHERE {
   join of triple statements after step 3
}
```

The keyword "DISTINCT" removes duplicate query solutions from the result set, leaving each remaining solution unique. The "*" is a SPARQL notation abbreviation that selects all of the variables in a query and, therefore, returns a column for each variable to the user.

Assembling a complete and coherent SPARQL query from the triples that were generated by the query generator is straightforward. Each class becomes a variable, each property also functions as property, and every specified value of a literal is wrapped into a "FILTER" statement. Additionally, domain and range of properties are checked for compliance and consistency, simultaneously ensuring correct variable types. As such, the following SPARQL query is assembled from the retrieved and joined triples for the question: "How big are the cities in Illinois?":

```
SELECT DISTINCT *
WHERE {
   ?City geo:hasPopulation ?Value .
   ?City rdf:type geo:City .
   ?City geo:isCityOf ?State .
   ?State rdf:type geo:State .
      FILTER REGEX (?State, "illinois") .
}
```

It can occur that more than one SPARQL query solution is proposed; this means that two or more queries achieved the same rating score in steps 1 to 3. Then, a simple empirically developed heuristic with two criteria filters the alternative SPARQL queries in order to retain only one query in the best case, or a small set of best-ranked queries. The heuristic is based on the following criteria:

1. All question words must appear in the query except those that are removed as stopwords. In other words, if there are question words left after performing steps 1 to 3, the corresponding SPARQL query is discarded.
2. Each triple that was retrieved by the query generator in the matching process (steps 1 to 3) must be included in the overall SPARQL query and joined with the other triples, thereby taking domain and range information into account. SPARQL queries including triples that were retrieved but could not be entirely joined are deleted.

If there are still several equivalent queries left, they are all executed by the Jena SPARQL engine ARQ, and the answer sets are displayed to the user in the different tabs of the user interface (as shown in Figure 4.3). Another possibility to deal with alternative solutions and, hence, multiple SPARQL translations for the same input question, would be to ask the user for clarifying feedback. The user could then choose the correct translation, thereby prompting the execution of only that SPARQL query. However, casual end-users would most likely be overly challenged by this process. We therefore decided in favor of executing each query and exhibiting the results in different tabs. The issue is not as serious, since the best SPARQL query solution is generally ranked first, but nevertheless, we implemented prompts that ask users for clarification when multiple query solutions emerge in our Querix interface (cf. Section 4.2). This has proven itself a useful and very welcome feature.

The query generator passes the SPARQL query to the next component of NLP-Reduce, the Jena SPARQL engine ARQ, which executes the query (or the queries) over the Jena model of the OWL knowledge base. Finally, the user interface displays the retrieved results to the user.

4.2 Querix

Similar to NLP-Reduce, Querix is a pattern-matching based, domain-independent NLI to Semantic Web data [Zumstein, 2006, Kaufmann et al., 2006]. In contrast to NLP-Reduce, however, Querix requires full English questions as its query language (see Figure 4.6). The approach is simple and does not use any complex semantics-based technologies. Compared to a full NLP search interface, Querix does not try to resolve natural language ambiguities, but asks the user for clarification in a dialog window if an ambiguity occurs in the input query. As such, the user acts the role of the druid Getafix (in German *Miraculix*), who is consulted by Asterix, Obelix and the other villagers whenever anything strange occurs (hence the name *Querix*). A strange event within Querix is an ambiguity in a query. The person composing a query benefits from the clarification dialog due to better retrieval results and, even more importantly, by being relieved from the cognitive burden of learning a formal query language [Chakrabarti, 2004].

The Querix system uses a parser to analyze the input query. From the parser's syntax tree, a query skeleton based on word categories is extracted, in which triple patterns are identified. By applying pattern matching algorithms that rely on the relationships that exist between the elements in a knowledge base, the triple patterns are then matched to the resources in the knowledge base. The matching and joining of the triples is controlled by domain and range information. From the joined triples, a SPARQL query is generated that can be executed by Jena's SPARQL Engine ARQ. Using WordNet, synonyms of the words in the query and the IDs in the knowledge base are included, providing an enhanced query language vocabulary and better matching.

If Querix encounters an ambiguity in a query, i.e., if several semantically different SPARQL queries could be generated, the clarification dialog of the interface pops up showing the different meanings that the system retrieved for the ambiguous element in a menu list. The user can then choose the intended meaning, and the interface executes the corresponding SPARQL query.

Querix lies between NLP-Reduce and the Ginseng interface (cf. Section 4.3) on the Formality Continuum, but more towards NLP-Reduce on the left and, hence, on the natural side of the continuum. Its query language is regarded as less natural and less flexible than the query language of NLP-Reduce, since only a limited set of natural language questions is actually allowed in Querix. However, the restriction refers only to the sentence beginnings. Once the beginning of a sentence is Querix-conform (i.e., starting with "Which," "What," "How many," "How much," "Give me," or "Does"), the rest of the sentence structure is free and uncontrolled, as long as it is a grammatically correct English sentence. The user interface is, therefore, sensitive to morphologically and syntactically ungrammatical input. Nevertheless, this limitation turned out to be absolutely unproblematic for even non-native English users as presented in Chapter 6.

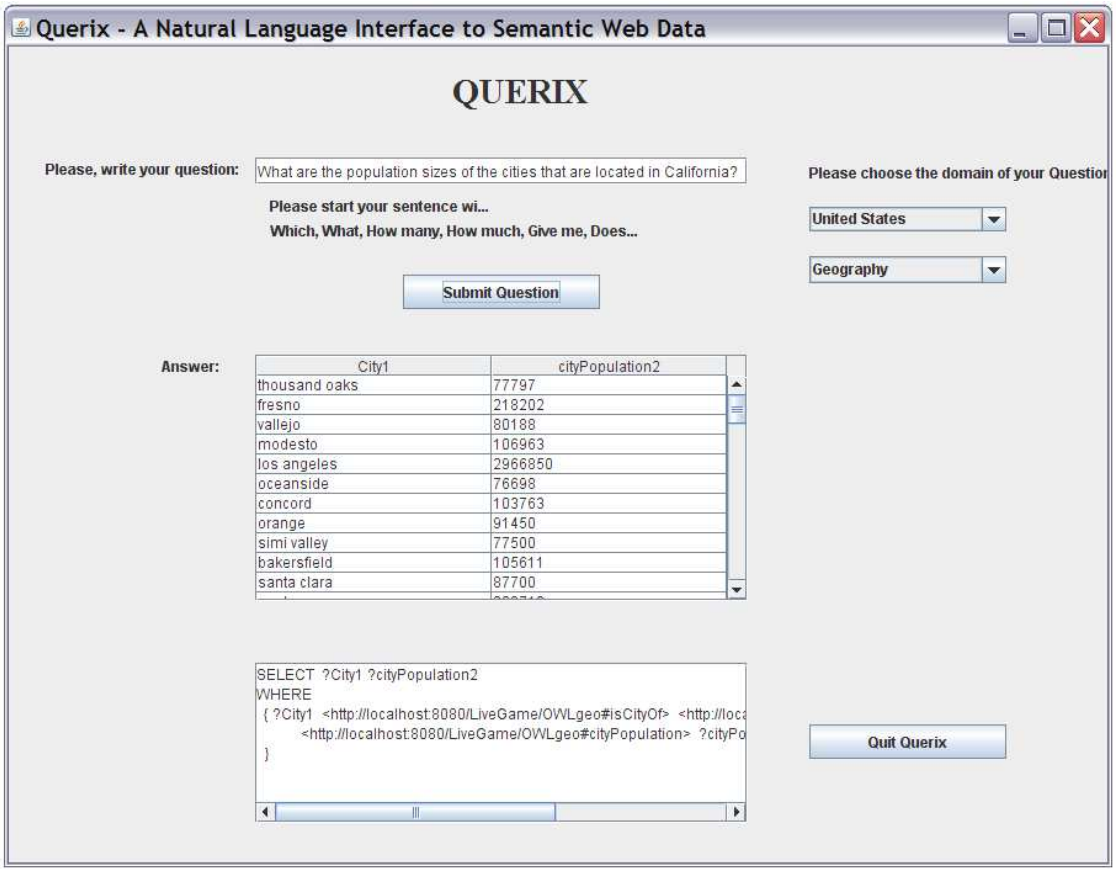


Figure 4.6: The Querix user interface after executing the question: "What are the population sizes of the cities that are located in California?"

Except for synonyms obtained from WordNet, Querix does not use any other vocabulary than the vocabulary given in a currently loaded knowledge base. Furthermore, it does not try to semantically interpret queries like most full-fledged NLIs do [Androutsopoulos et al., 1995]. As such, Querix is not an "intelligent" system that interprets and understands the input queries; it requires a slightly controlled natural language query language, employs a small set of simple NLP tools, and consults the user upon reaching its limitations. But Querix does, in contrast to NLP-Reduce, make use of the syntactical structure of input questions, which should enable a better matching between the queries and terms existing in the knowledge base. Querix's almost unlimited natural language query language not only increases the risk that a query does not lead to a correct SPARQL translation and, therefore, to a correct answer, but it also increases the habitability problem, albeit less immediately than NLP-Reduce. The benefits, on the other hand, are obvious: a simple, fully portable NLI as well as an interface with a natu-

ral query language providing access to knowledge bases for casual end-users not familiar with the formality of the Semantic Web.

In the following sections, we will present the Querix user interface by introducing the users experience of querying with Querix. We will then discuss its technical design, which features the most complicated configuration of components among our four interfaces. After the technical overview, we focus on three parts of Querix: the query analyzing component, which prepares an input question for the matching with the knowledge base, the matching center that actually conducts the matching by applying an empirically developed triple-matching heuristic, and the problem manager, which is responsible for handling query ambiguities and asking the user for their resolution.

4.2.1 Querying in Querix—The User Experience

Basing our actions on the assumption that NLIs provide a useful means for querying access to Semantic Web data, we implemented an interface allowing a limited set of full English sentences. As such, Querix pursues two goals. First, it hides the formality of an ontology-based knowledge base as well as the executable query language from end-users by offering an intuitive and familiar way of query formulation. Second, it avoids a tedious, complex, and domain-tailored system configuration. The latter goal is attained by slightly limiting the natural language query language to a set of sentences that must begin with one of the following question or imperative sentence beginnings:

- "Which ..."
- "What ..."
- "How many ..."
- "How much ..."
- "Give me ..."
- "Does ..."

The restriction was supported by the literature's observation that people tend to use a simple, limited language [Chakrabarti, 2004, Hersh et al., 1999, Malhotra, 1975] [Spink et al., 2001]; in particular, they use simple questions when interacting with a system as opposed to conversing with other people [Dittenbach et al., 2003] [Linckels and Meinel, 2006].

In order to be suitable for casual end-users, the user interface of Querix is simple, requiring almost no training. At the top, there is a free text entry field where the user

can enter the query (cf. Figure 4.7). Punctuation marks are not required except for the question mark or the full stop at the end of the query. Either of them is mandatory, depending on whether the sentence is of interrogative or imperative type such that the Stanford Parser [Klein and Manning, 2002] knows when a sentence is finished. The interface is not case-sensitive. After typing a query, the user can click on the button "Submit Question" in order to start Querix processing and answering the query.

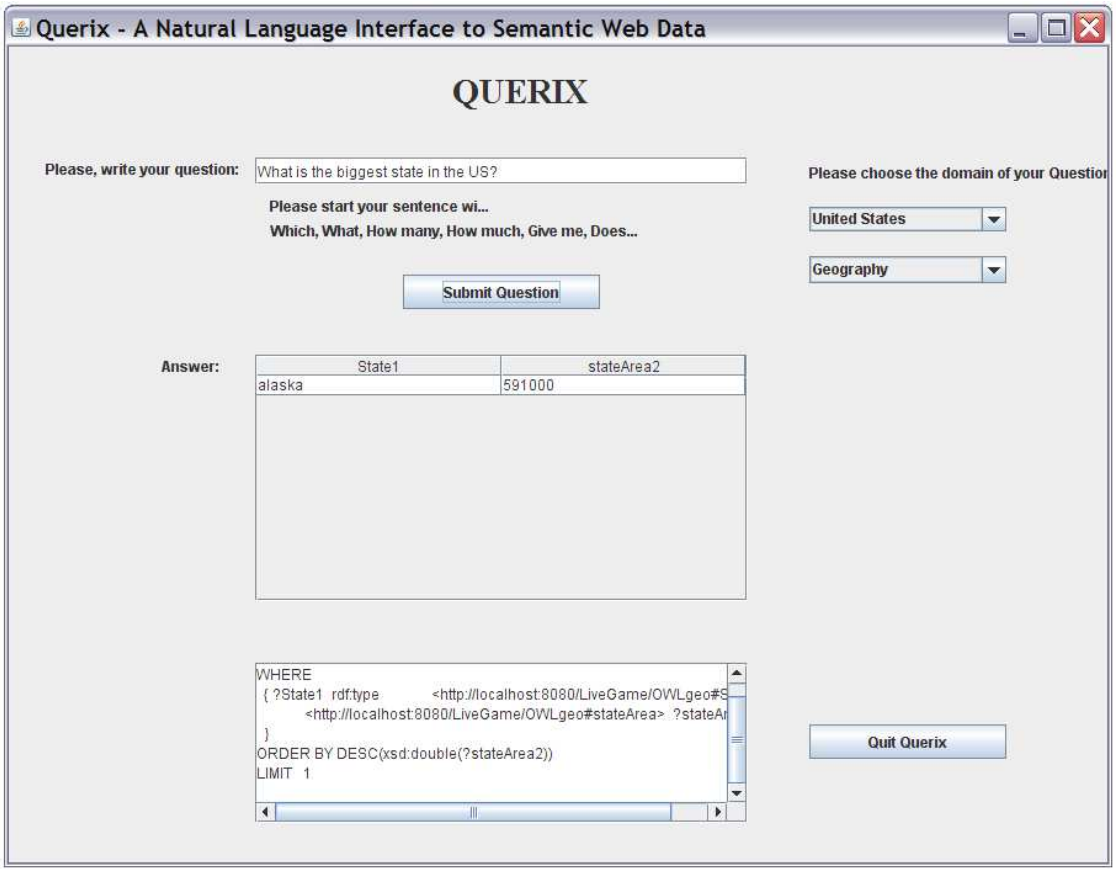


Figure 4.7: The Querix user interface showing the question "What is the biggest state in the US?", the returned result (in the middle), and the SPARQL representation on the bottom of the interface. On the right side, a user can specify the knowledge base to be loaded and queried.

Below the query entry field is the answer field displaying results in the form of a table. Similar to NLP-Reduce, a column headed by the name of each variable that occurs in the generated SPARQL query is displayed. The display of answers to questions beginning with "How many ..." or "How much ..." differs, as illustrated in Figure 4.8. We hypothesized that this kind of answer would be suitable for casual end-users, and our supposition was confirmed by the usability study (presented in Chapter 6), which

pointed out that the feature is, indeed, convenient.

The field on the bottom of the interface shows the generated SPARQL query. If Querix cannot generate a translation usually due to the presence of unknown terms, it outputs "No solution found. Please check your query." in the answer field.

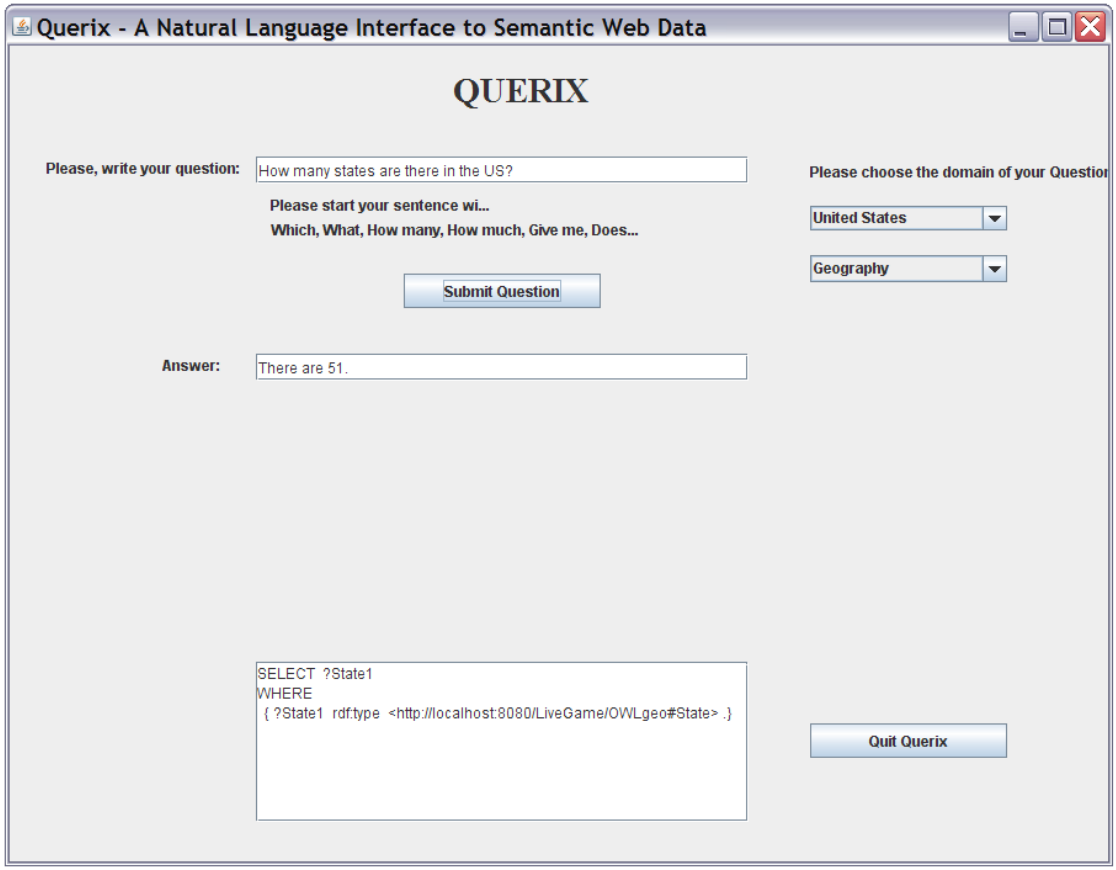


Figure 4.8: The Querix user interface returning the natural language answer "There are 51." to the question "How many states are there in the US?".

On the top right side of the interface, one can specify which domain, i.e, which knowledge base, should be loaded and prepared for querying. The specification is a multiple-step process, in which a general domain is first selected and further circumscribed by a more specific area in a follow-up step. A user could, for example, be interested in finding a restaurant in the US. To do so, he/she would choose "Society" instead of Geography and then "Restaurants" depending on the available knowledge bases. The menu lists can be changed arbitrarily and adapted by a system administrator, even according to an ontology hierarchy describing such fields of interest.

Due to the natural and almost free query language, it can occur that Querix encounters an ambiguity in a query meaning that several possible, semantically different SPARQL queries can be generated for a single natural language query. In this case, the clarification dialog of the interface opens in what is called the *AskBox* window (see Figure 4.9). The AskBox asks the user for clarification by presenting all properties that could be related to an ambiguous query expression to the user. The user then selects the intended meaning, i.e., the property best representing the meaning, from a menu list. After receiving clarifaction, Querix executes the corresponding SPARQL query. Consider, for example, the query "What is the biggest state in the US?", in which the word "biggest" is ambiguous as it can refer to the properties `hasStatePopulation`, `hasStatePopulationDensity`, and `hasStateArea` of a knowledge base containing geographical information. If the user selects `hasStatePopulation`, the answer to the query is "California;" if `stateArea` is selected, the answer Querix returns is different, namely "Alaska."

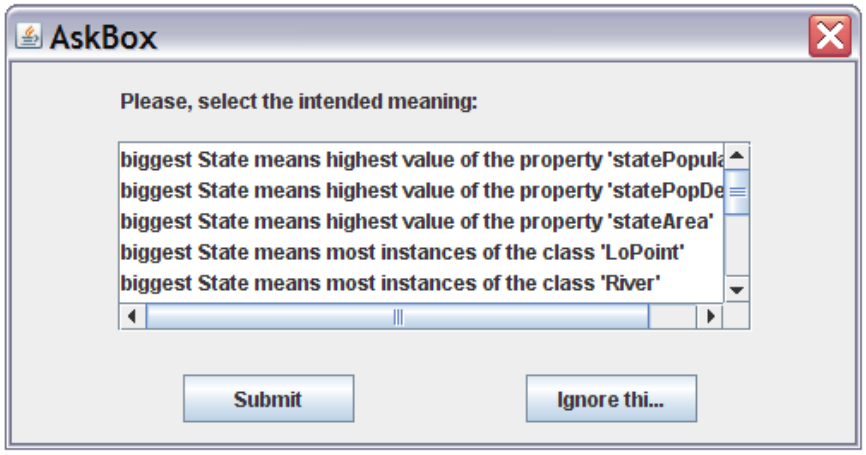


Figure 4.9: After detecting that more than one possible SPARQL query can be generated for the question "What is the biggest state in the US?", the AskBox of Querix opens in a pop-up window and offers the possible meanings of "biggest" to the user for selection.

Querix's approach to handling natural language ambiguities is not only straightforward, but it also avoids the implementation or application of a complex ambiguity resolution algorithm [Linckels and Meinel, 2006]. As such, besides the restriction of the query language to a fixed set of sentence beginnings, the clarification dialog helps to achieve our goal of avoiding an overall complicated and tedious system configuration. Apart from that, the clarification dialog turned out to be an extremely convenient feature that immensely boosted the popularity of Querix among the participants taking part in our usability study evaluation (see Chapter 6).

4.2.2 Technical Overview

The Querix system consists of four main parts: a user interface, a query analyzer, a matching center, a problem (or ambiguity) center, and an ontology access layer. Each part in turn comprises several other components, as illustrated in Figure 4.10. The architecture of Querix is, in fact, similar to that of NLP-Reduce, particularly with regard to the ontology access layer and the matching component, but also more complex, since Querix employs a syntax parser plus WordNet in order to preprocess a user query, a problem manager facilitating the clarification dialog with the user, and a slightly more sophisticated matching algorithm that also considers a query's syntactical information besides the ontology-based information.

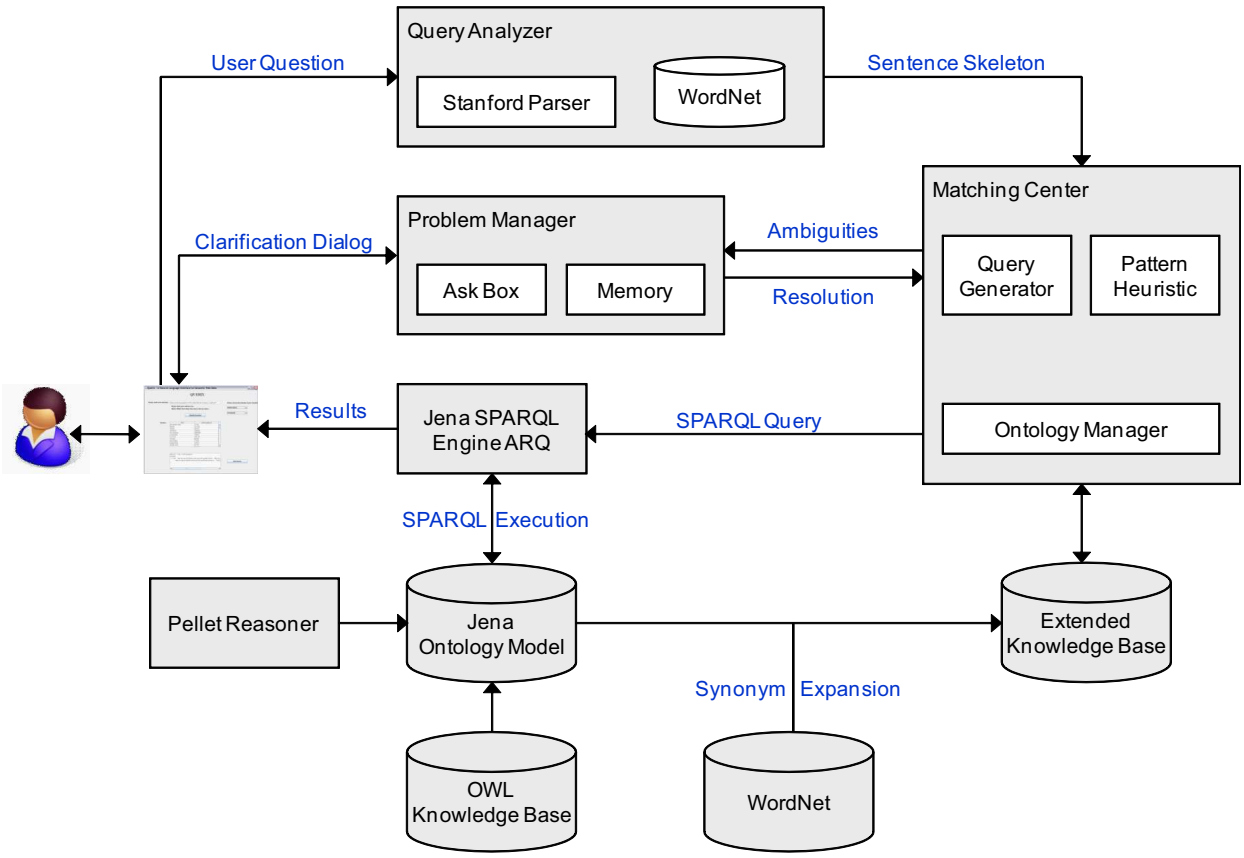


Figure 4.10: The Querix architecture exhibiting a query analyzer including the Stanford Parser and WordNet, a matching center that matches a query skeleton to the synonym-enhanced knowledge base, which applies a word-category-based triple matching heuristic, and a problem center that facilitates the display of ambiguities to the user in the ask box window such that the user can choose the intended meaning. The Jena framework is used as ontology-access layer and its ARQ as SPARQL execution engine.

As mentioned above, the *user interface* allows people to enter full natural language queries that are restricted with regard to their sentence beginnings, and then choose the ontology to be queried. After executing a query, it displays the results and the SPARQL query it generated for the query to the user.

The *query analyzer* employs two auxiliary components. The first component is the Stanford Parser [Klein and Manning, 2002], which provides a syntax tree for the natural language query. From this syntax tree, the query analyzer extracts the sequence of the word categories "Noun," "Verb," "Wh-word," "Preposition," and "Conjunction." Based on the extracted word categories, a query skeleton is generated. The second query analyzer component is WordNet, which provides synonyms for all nouns and verbs in the query's parse tree. As such, each query word belonging to one of the extracted word categories is stored with its word form, while nouns and verbs are additionally stored with their synonyms. This query skeleton sequence is subsequently forwarded to the matching center component.

The *matching center* is the core component of Querix. It attempts to match the query skeleton with the synonym-enhanced triples of the knowledge base. When a knowledge base is chosen and read into Querix, the knowledge base's RDF triples are first loaded into a Jena Model, with the Pellet reasoner attached to it inferring all implicitly defined triples. The matching center's *ontology manager* then enhances the resources' IDs of the Jena Model by obtaining synonyms from WordNet; more precisely, for each noun and verb that occurs in a triple of the Jena Model, the associated synonyms are added. The synonym-enhanced triples, including their domain and range specifications, are finally stored as extended knowledge base.

By applying a small set of heuristic patterns, query skeletons can be matched to the triples in the extended knowledge base. The matching center essentially identifies overlapping subject–property–object patterns in the query skeleton and matches them to those triples that are retrieved from the extended knowledge base by searching for triples that include one of the nouns or verbs from the query skeleton. The resulting matches are joined according to the domain and range specifications of the triples. Eventually, the matching center's *query generator* composes SPARQL statements from the joined triples and passes the complete SPARQL query to the Jena ARQ for execution.

If Querix encounters ambiguities, i.e., several different solutions to a single question, its *problem manager* is invoked. The problem manager consists of two components, the *AskBox* and the *memory*. The AskBox facilitates consultation of the user in the case of ambiguities by showing a menu from which the user can choose the meaning she/he intended. In this way, the system retrieves the correct resolution for the ambiguity. The possible meanings offered by Querix are based on different possible triples that are identified by the matching center. After receiving clarification, Querix can execute the corresponding SPARQL query and display the requested result in the user interface. The

problem manager also comprises a memory, in which an ambiguity, its possible meanings, and the meaning selected by a user are stored. As such, the chosen resolution can be presented as the first of the list of possible meanings, therefore, supporting the user in the clarification process.

In order to provide a more precise and comprehensive description of how Querix works, we will further explain the functionality of the query analyzer, the matching center, and the problem manager in the following three sections. Each step will be demonstrated with the following running example question: "What are the population sizes of the cities that are located in California?".

4.2.3 The Query Analyzer

The query analyzer of Querix preprocesses and analyzes queries entered by users in order to prepare them for matching with the triples of a knowledge base. It performs two steps, thereby employing two auxiliary components. The first component is the Stanford Parser [Klein and Manning, 2002], which provides a syntax tree for the natural language query. The Stanford Parser is open-source (`http://nlp.stanford.edu/software/lex-parser.shtml`), unbeatable in terms of speed, and implemented in Java. These attributes made it favorable over other available parsers such as the Charniak Parser [Charniak, 2000], which is known to be the best-performing of the current natural language parsers [Hempelmann et al., 2005]. However, the Stanford Parser's performance quality is only marginally poorer than the Charniak's. For the example question, "What are the population sizes of the cities that are located in California?", the Stanford Parser returns the syntax tree as depicted in Figure 4.11.

From the syntactical constituents of the parse tree delivered by the Stanford Parser, the query analyzer first extracts the sequence of the word categories *noun* (N), *verb* (V), *preposition* (P), *wh-Word* (Q), and *conjunction* (C). Based on these extracted word categories, a query skeleton is generated. Consider our example: "What are the population sizes of the cities that are located in California?". According to the query analyzer, its query skeleton is: Q-V-N-P-N-Q-V-P-N. Each word category of the sentence's skeleton is stored together with its word form. As such, the information presented in Table 4.1 is stored by the query analyzer after performing the first step.

The second query analyzer component is WordNet, which provides synonyms for all nouns and verbs in the query's parse tree. We implemented a cost function in order to obtain only the most appropriate synonyms, as WordNet usually suggests too many. Furthermore, since WordNet was not designed to handle multi-word terms, we processed each word of a multi-word term individually,. The union of all obtained synonyms is then stored with the corresponding word form.

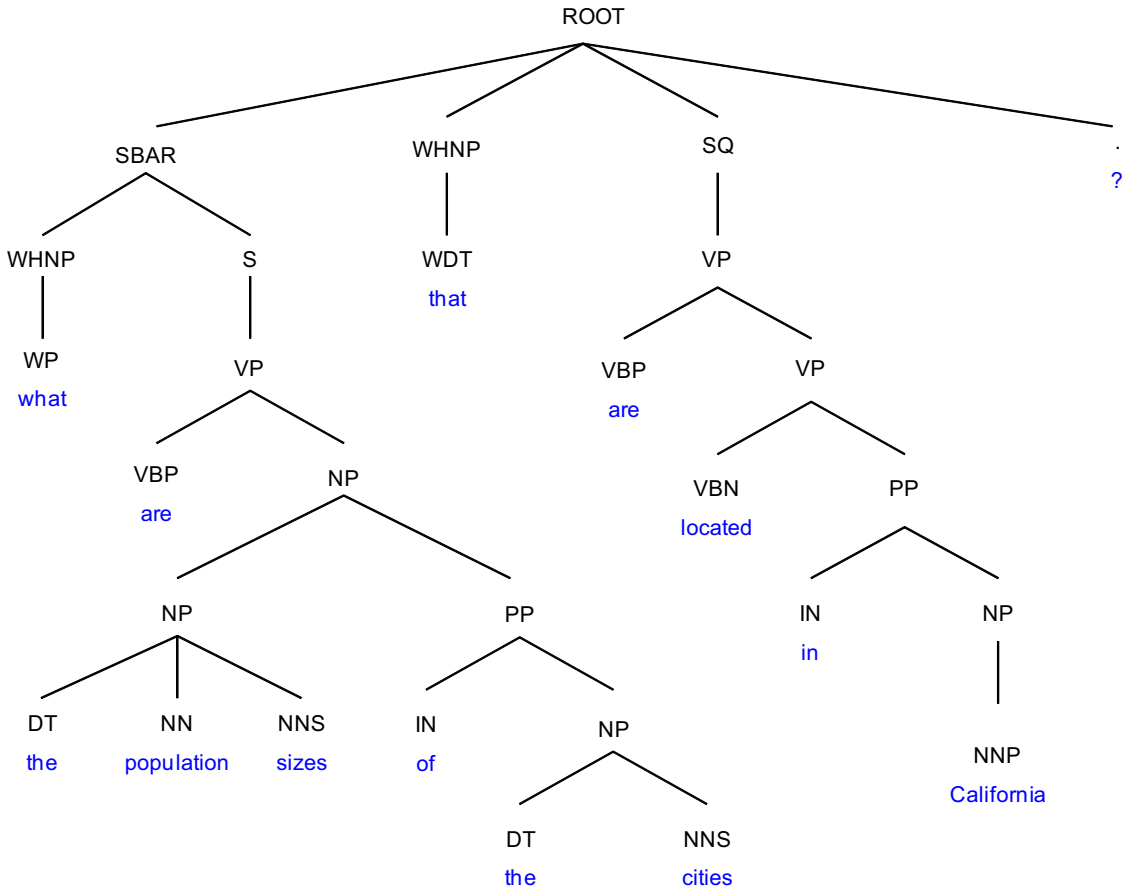


Figure 4.11: The parse tree (in pretty printing) generated by the Stanford Parser for the question: "What are the population sizes of the cities that are located in California?".

Q	V	N	P	N	Q	V	P	N
what	are	population sizes	of	cities	that	are located	in	California

Table 4.1: For the question "What are the population sizes of the cities that are located in California?", the query analyzer extracts the word categories *noun* (N), *verb* (V), *preposition* (P), *wh-word* (Q), and *conjunction* (C) from the questions's syntax tree and stores the word category together with the word form that occurred in the query.

After consulting WordNet, the information that is shown in Table 4.2 is stored by the query analyzer and forwarded to the next component of Querix, the matching center.

Q what	V are	N population sizes	P of	N cities	Q that	V are located	P in	N California
	be exist	inhabitant citizen number magnitude measurement		town metropolis urban center municipal		situate place site settle		CA Golden State

Table 4.2: After retrieving synonyms for all nouns and verbs of the question "What are the population sizes of the cities that are located in California?", the query analyzer stores the synonyms together with the word forms and the word categories. This information is then forwarded to the matching center.

4.2.4 The Matching Center

Querix's matching center accomplishes the matching of a query skeleton with the synonym-enhanced triples in the knowledge base. For the example query (and, of course, for every query), the following steps are performed by the matching center:

Step 1. The matching center receives the query skeleton with the information (as seen in Table 4.2) from the query analyzer and first attempts to match the extracted query skeleton with a small set of heuristic patterns. In other words, it tries, based on word category sequences, to identify subject–property–object triple patterns in the skeleton. The heuristic consists of only four patterns and is shown in Table 4.3.

Pattern Name	Form	Short Form	Example
preposition pattern	noun–preposition–noun	N–P–N	capital of state
verb pattern	noun–verb–noun	N–V–N	state borders state
wh pattern	noun–wh-word–verb–[preposition]–noun	N–Q–V–[P]–N	state that has city city that lies in state
conjunction pattern	noun–conjunction–noun	N–C–N	mountains and rivers

Table 4.3: The set of heuristic patterns that the matching center of Querix uses in order to detect triple patterns in a query skeleton.

When defining the patterns, our aim was to minimize as much as possible the number of patterns such that a few basic patterns cover as many cases as possible in order to achieve an intensional definition of the patterns rather than an extensional specification that lists every possible pattern (as suggested in the literature [Witten and Eibe, 2005]). We ended up with four patterns, one of which possesses

an optional word category, i.e., the wh-pattern with an optional preposition, thus leaving us with essentially five word category sequence patterns.

The matching center matches a query skeleton to these patterns, thereby starting from the skeleton's end, i.e., from the right side of the sentence. We observed that sentence beginnings offer a great variability with regard to lexical and syntactical expressions. Hence, identifying patterns using a sentence's end proved a successful approach. Valid patterns in the query skeleton have to overlap with regard to their first or last word category, and, when viewing the forms of the patterns, it is obvious that overlapping elements are always nouns. This linking of nouns mirrors the nature of subject–property–object triples in ontologies or subject–predicate–object structures in linguistic sentences, in which nouns or nominal phrases typically occupy the subject and object positions.

In our example query skeleton, two patterns can be matched starting from the skeleton's end, as depicted in Figure 4.12.

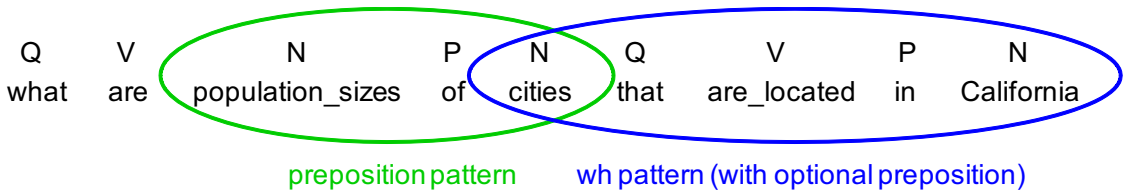


Figure 4.12: Basic query patterns identified in the query skeleton of the question: "What are the population sizes of the cities that are located in California?".

If no more patterns can be found, the matching center repeats the same pattern matching procedure for the remaining categories towards the skeleton's beginning, but, this time, each noun category (N) can be replaced by a wh-category (Q). In this way, the linguistic notion that all noun constituents (or rather nominal phrases) can be replaced by pronouns and, moreover, that replacing and fronting a constituent in a main clause by a wh-word has the effect of turning it into a question, is reflected. Since Querix allows questions starting with "Which ...," "What ...," or "How ...," our four heuristic patterns can still match those question beginnings by accepting a wh-word category in place of a noun category. Figure 4.13 shows the result after the second matching iteration with the heuristic patterns for the example query. The figure illustrates that the remaining word categories towards the skeleton's beginnings are also successfully matched, and matching step 1 is completed.

Step 2. In the next step, the matching center searches for all matches between the synonym-enhanced nouns and verbs of the input query with the resources and their

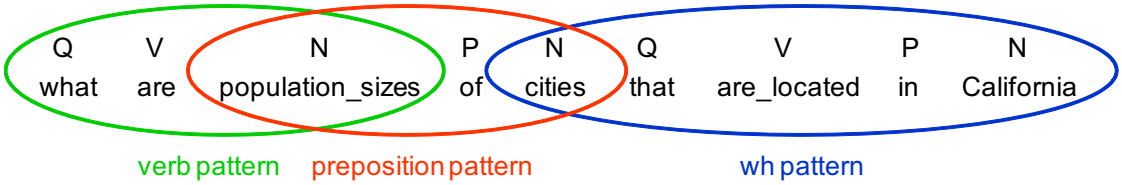


Figure 4.13: All query patterns identified in the query skeleton of the question: "What are the population sizes of the cities that are located in California?" after the second matching iteration that also considers wh-words replacing nouns.

synonyms in the preprocessed and extended knowledge base. Recall that, for every verb and noun occurring in the input questions, their corresponding synonyms were stored (see Table 4.2). Additionally, the ontology manager obtained synonyms for each noun and verb appearing in a triple of the Jena Model after loading a knowledge base into Querix. These synonym-enhanced triples, including their domain and range specifications, constitute the contents of the extended knowledge base (see Figure 4.10). The matching center can now make use of both information repositories, and matches each noun or associated synonym of a query skeleton with the nouns or synonyms stored in the extended knowledge base. It also does the same for verbs. As such, query words are related to resources and triples in the knowledge base. Each possible match is stored, again including domain and range information.

For our example question, 'What are the population sizes of the cities that are located in California?", and its generated query skeleton, the following relevant triples can be found in the extended knowledge base:

are:
```
domain=[geo:city] geo:isCityOf range=[geo:state]
```

population sizes:
```
domain=[geo:state,geo:city] geo:hasPopulation
   range=[xml:float]
```

cities:
```
domain=[geo:city] geo:isCityOf range=[geo:state]
```

are located:
```
domain=[geo:city] geo:isLocatedIn range=[geo:state]
```

California:
```
domain=[geo:city] geo:isLocatedIn geo:california
```

Obviously, we did not list every triple that was retrieved for the nouns, verbs, and their synonyms, since a few dozens of triples would have had to be given. We limited ourselves to those that will be used in the matching center's step 3.

Step 3. Finally, the triple patterns identified by step 1 and the resources found by step 2 are matched. This matching, step 3, is enabled by storing all the information that is shown in Table 4.2 together with the triple patterns as displayed in Figure 4.13, and is controlled by the domain and range restrictions of the triples found in matching step 2. As such, the following matchings are yielded:

wh-pattern "cities that are located in California":
```
domain=[geo:city] geo:isLocatedIn range=[geo:state],
domain=[geo:city] geo:isLocatedIn geo:california
```

preposition pattern "population sizes of cities":
```
domain=[geo:state,geo:city] geo:hasPopulation
   range=[xml:float]
```

verb pattern "what are population sizes":
```
domain=[geo:state,geo:city] hasPopulation range=[xml:float]
```

Step 4. After identifying all possible triples in the sentence skeleton and combining them to the knowledge base's resources, the *query generator* of Querix's matching center attempts to join the triple statements stored after matching step 3 in order to generate, as a last step, a corresponding SPARQL query. The joining is controlled by the triples' domain and range information, and by the order of the word category patterns found in the query skeleton—in particular by the overlapping noun categories. Hence, the following temporary join of triples is found for our running example:

```
domain=[geo:city] geo:hasPopulation range=[xml:float],
domain=[geo:city] geo:hasPopulation range=[xml:float],
```

```
domain=[geo:city] geo:isLocatedIn range=[geo:state],
domain=[geo:city] geo:isLocatedIn geo:california.
```

The query generator next reduces the join by removing redundant triples while still complying with the domain and range specifications. Only two triple statements remain after the removal:

```
domain=[geo:city] geo:hasPopulation range=[xml:float],
domain=[geo:city] geo:isLocatedIn geo:california.
```

From the joined triples, a syntactically correct and coherent SPARQL query is composed that completes the processes performed by the matching center:

```
SELECT ?city ?population
WHERE {
   ?city geo:hasPopulation ?population .
   ?city rdf:type geo:City .
   ?city geo:isLocatedIn ?State .
   ?state rdf:type geo:State .
      FILTER REGEX (?state, "california") .
}
```

Finally, the result is displayed to the user. The top answers to the query and, hence, to the question "What are the population sizes of the cities that are located in California?" are presented in Figure 4.6 on page 62.

4.2.5 The Problem Manager

The problem manager is the third component that we developed and implemented in the Querix system. Its basic duty is to handle ambiguities, i.e., several different solutions to the same query are found, and to ask the user for clarification if an ambiguity occurs in a query. The problem manager applies in two cases: (1) if several different SPARQL queries can be generated for the same question generally, and (2) if an ambiguous adjective occurs in a question particularly. In both cases, the user is asked to provide additional information in order to resolve the ambiguity.

If more than one SPARQL query containing different properties, can be generated for the same question, an ambiguity of the first case occurs. The AskBox then initializes a

dialog with the user, in which those different properties are offered in a menu for selection (as shown in Figure 4.9 on page 66). The user chooses the intended meaning and causes the system to retrieve the correct answer. The noun "population," for example, can refer to two properties `geo:hasPopulation` and `geo:hasPopulationDensity` in an ontology containing geographical information. A natural language question such as "What is Nevada's population?" finds both properties in the matching process and shows them in the menu of the AskBox. The user selects the property best matching his/her information need.

The second case of ambiguity appears if an adjective is used in a query. In the above section describing the matching center, adjectives have entirely been ignored. Nevertheless, they occur in queries and have to be translated into adequate SPARQL statements. The Stanford Parser identifies and tags adjectives depending on their forms as follows:

`big/JJ`	`JJ` standing for "adjective"
`bigger/JJR`	`JJR` standing for "adjective, comparative"
`biggest/JJS`	`JJS` standing for "adjective, superlative"

Basing on the parser's tagging information, the query analyzer can easily recognize adjectives and pass them to the matching center. The matching center ignores them during the matching process for the rest of a query, but also retrieves synonyms from WordNet for the adjective. It then tries to match the adjective and its synonyms to the triples in the knowledge base separately, in a manner similar to matching step 2. If these adjective-matching triples are included in the triples that were found by processing the rest of the query, than the triples matching the adjective are discarded. If the matching of the adjective adds additional triples to the ones obtained from the remaining question words, then those triples are forwarded to the problem manager and, consequently, offered to the user for clarification.

An adjective possessing a comparative or superlative form is always passed to the problem center, since an appropriate answer to a question containing these forms requires additional filtering of the result set. The problem manager then inserts the necessary SPARQL solution modifier statements into a query. Consider, for example, the question "What is the biggest state in the US?" (which is displayed in Figure 4.7). The word "biggest" is ambiguous as it can, according to WordNet, refer to the properties `geo:hasStatePopulation`, `geo:hasStatePopulationDensity`, and `geo:hasStateArea`. If the user selects one of the meanings, all answers except the one that has the highest value for the chosen datatype property must be filtered out. The filtering is established by the SPARQL solution modifier `ORDER BY` clause, which determines the order of a solution sequence. If the user selects `geo:hasStatePopulation`, for example, Querix returns the answer "California." If `geo:stateArea` is selected, the

answer by Querix is different, namely "Alaska."

Comparative adjectives are handled similarly. The adjective "longer" of a question such as "Which rivers are longer than the Rio Grande?" is forwarded to the problem center in order that potential ambiguities can be presented to the user for clarification and that the generated SPARQL query is augmented with the SPARQL solution modifiers:

```
ORDER BY,
OFFSET,
LIMIT.
```

`ORDER BY` establishes the order of a solution sequence, `OFFSET` causes solutions to begin after the specified number of solutions, and the `LIMIT` clause puts an upper boundary on the number of solutions returned [Prud'hommeaux and Seaborne, 2007]. The three modifiers are sufficient to adequately answer questions containing a comparative adjective form, such as the question asking for rivers that are longer than the Rio Grande: "Mississippi, Missouri."

Apart from the AskBox that enables the clarification dialog with the user, the problem manager possesses a second component, the *memory*. The task of the memory is to store an ambiguous query, the different properties that were associated with the query and offered in a menu for selection, and the choice of the user. As such, the information can be reused if the same query is entered again. The chosen property will be presented on top of the list of all possible properties (meanings), therefore offering some support to the user in the clarification process.

Supported by Chakrabartis assertion that users would express their need for information in more detail, thereby receiving better search results [Chakrabarti, 2004], we decided to implement the AskBox mechanism, which rewards the user by avoiding unnecessary or incorrect answers. We intended to forgo a complex ambiguity resolution algorithm in the sense of Linckels and Meinel [Linckels and Meinel, 2006]. Unexpectedly, the clarification dialog asking for hints did not annoy the users participating in our usability study; on the contrary, it lead to the impression that the search interface sought to understand them—they found this pleasing. Furthermore, it increased user control for the natural and, hence, imprecise query language in a convenient manner.

Akin to the simple design of the problem manager and the perhaps rather evasive solution for ambiguity treatment, Querix generally deliberately forgoes any complicated processing techniques. It does not exploit sophisticated logic-based or semantic techniques as typical full-fledged NLP systems do. We, therefore, restricted Querix with regard to sentence beginnings and ambiguity resolution in return. The dependencies

between the words and phrases in the queries are identified by applying only two auxiliary NLP tools and pattern matching algorithms, both of which rely on the relationships that exist between the elements in the queried knowledge base. The approach, therefore, highly depends on the quality and choice of vocabulary of the ontology. We have seen that this weakness can also be a system's major strength, such as with NLP-Reduce's or Querix's, as both search interfaces do not need any adaptation for new knowledge bases and, hence, are portable.

4.3 Ginseng

The third NLI we developed is Ginseng, a *guided input natural language search engine* for Semantic Web knowledge bases [Kaiser, 2004, Bernstein et al., 2005b] [Bernstein et al., 2006, Bernstein and Kaufmann, 2006]. Ginseng belongs to the family of menu-guided NLIs and provides a quasi natural language querying access to any ontology-based knowledge base, as long as the knowledge base is expressed in OWL. It relies on a simple question grammar, which is dynamically extended according to the knowledge base. The extended grammar can be used to parse queries which strongly resemble plain English. The affinity of the Ginseng query language to plain English can be increased if meaningful names (i.e., IDs) for classes and their properties in the OWL knowledge base are chosen and additional synonym information is added into the ontology.

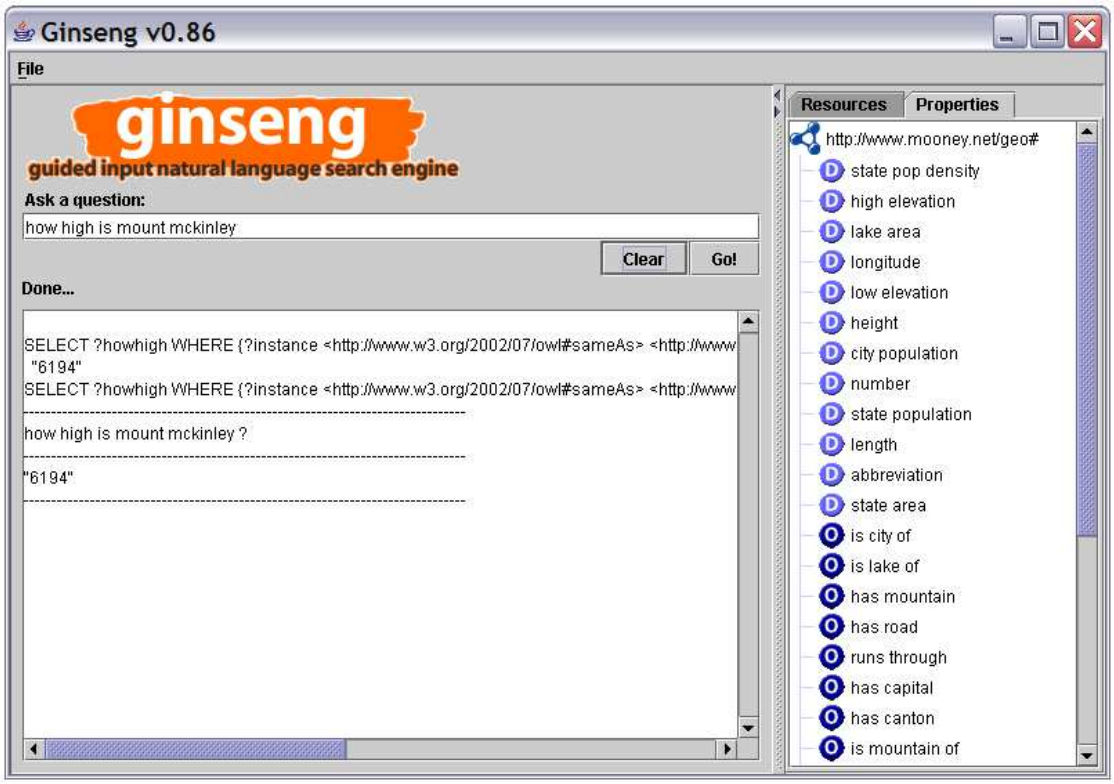


Figure 4.14: The Ginseng user interface after executing the query: "How high is Mount McKinley?"

When the user enters queries, an incremental parser relies on the grammar to constantly check the user's entries in order to (1) propose possible continuations of the query and (2) prevent entries that would not be grammatical. Once a query is completed,

Ginseng uses additional query construction information in the grammar to translate the English entry to SPARQL statements, and then passes them to Jena for execution. The SPARQL query as well as the answer set are then displayed to the user as shown in Figure 4.14. As such, Ginseng provides a guided input natural language search engine for Semantic Web data.

The main difference between Ginseng and full-fledged NLI [Androutsopoulos et al., 1995] is that Ginseng does not use any predefined vocabulary beyond the vocabulary given in the knowledge base itself and in the question grammar. Moreover, similar to NLP-Reduce and Querix, it does not try to interpret the queries (logically or syntactically). Instead, Ginseng "only knows" the vocabulary that is being defined by the currently loaded knowledge bases. The querying vocabulary is closed, and the user must follow it. This can limit the user's possibilities in general, but ensures that all queries make sense in the context of the loaded knowledge bases—every query leads to a properly translatable result. As such, any linguistic preprocessing to improve recall (e.g., stemming, parsing) can be avoided by limiting the user to what the system "understands." Further alleviating this limitation, it is important to note that the vocabulary grows with every additionally loaded knowledge base, though users have signaled that they prefer to load only one knowledge base at a time.

We will now first introduce the Ginseng user interface by illustrating the user's querying experience. Next, we will discuss the technical design of Ginseng, including the structure of the automatically extended grammar and how it is used by the incremental parser, as well as the functionality of Ginseng's grammar compiler.

4.3.1 Querying in Ginseng—The User Experience

Ginseng allows users to query any OWL knowledge base using a guided natural language. As shown in Figure 4.15, queries are entered in English into a freeform entry field. When the user starts typing, the system predicts the possible completions of what the user enters in a manner similar to the completion suggestions in Unix shells or the "code assist" and "intellisense" in integrated development environments. Based on the grammar, the system's incremental parser offers the possible completions of the entry by presenting the user with choice pop-up boxes. These pop-up menus offer suggestions on how to complete a current word or on what the next word might be. Obviously, the possible choices are reduced as the user continues to type. By replacing the current pop-up with a *Fisheye* pop-up menu [Bederson, 2000], even very large knowledge bases could be handled by Ginseng.

For example, when typing the letter "c" within the middle of a query, the interface proposes all the possible continuations of the query that commence with "c." After con-

Figure 4.15: The Ginseng user interface showing a query completion pop-up choice menu when typing a "c" in the middle of a query.

tinuing with "a," Ginseng limits the choice to "california," "cambridge," "camden," etc. (cf. Figure 4.15). When a new word is started, all the possible words are shown in the pop-up. Users can navigate the pop-up with the arrow keys or with the mouse and choose one of the proposed options. Any entries not in the list are unacceptable to the grammar. Therefore, Ginseng does not allow their entry and (optionally) beeps an error message. The user is thus guided through the set of possible words and queries via the pop-up, which prevents those considered unacceptable by the grammar. When the query is completed, Ginseng translates the query to SPARQL statements, executes the SPARQL query against the ontology model using Jena, and displays the result set to the user. Figure 4.16 shows the query, the generated SPARQL statements, and the result of the query "What are the capitals of the states that border Massachusetts?".

The graph representation on the right side of the Ginseng user interface offers an overview of the classes, properties, and instances of the currently loaded knowledge base as well as an easy editing function. In Figure 4.15, all classes of a knowledge base with geographic information are shown designated by the dark yellow circles with a "C."

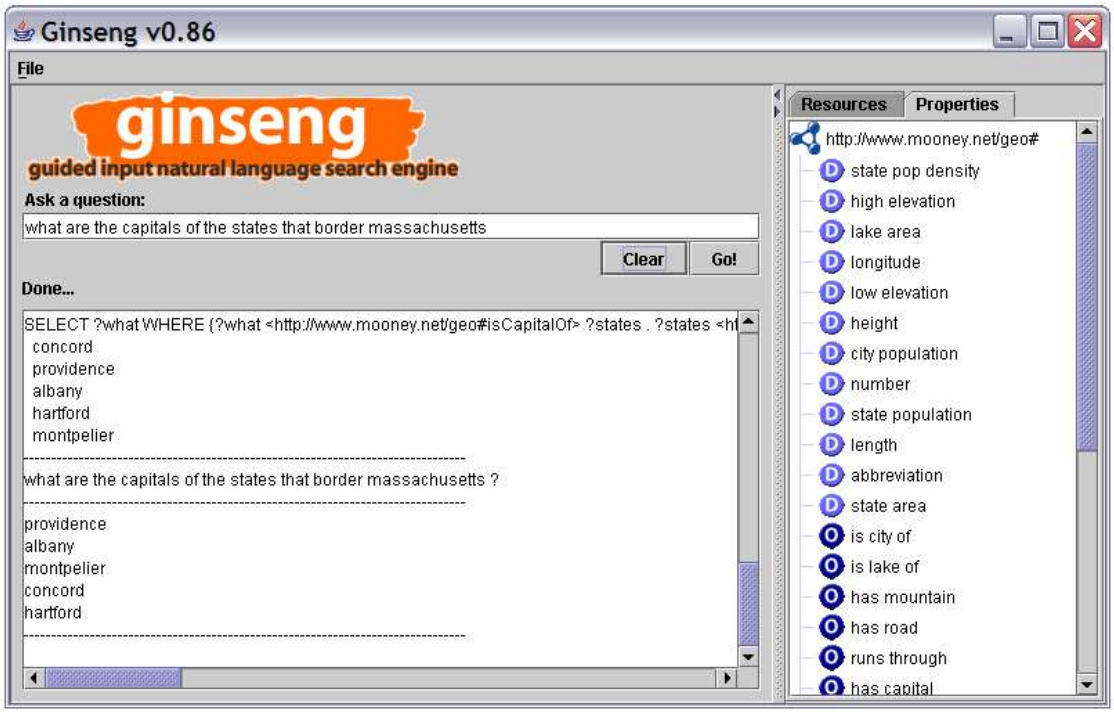


Figure 4.16: The Ginseng user interface showing the query "What are the capitals of the states that border Massachusetts?", the generated SPARQL statements, and the result of the query.

When a class is clicked on, its member instances are presented as shown in Figure 4.17; they have red circles with an "I" to identify them as instances.

When the tab "Properties" is clicked, all properties of a currently loaded knowledge base, here subdivided into datatype properties (light blue circles with a "D") and object properties (dark blue circles with an "O") are presented as in Figure 4.16. By double-clicking on an element, an edit window is opened where the user can add, change, or delete elements, values, etc. Double-clicking on the property "lakeDepth" in the property tree, for example, opens an edit window showing the specification possibilities of the property (see Figure 4.18 on page 84). The type of the property, its domain and range can be specified if needed. Similarly, classes and instances can be added, removed, or edited. More information on Ginseng's ontology editor extension can be found in Bernstein and Kaufmann [Bernstein and Kaufmann, 2006].

Androutsopoulos *et al.* and Thompson *et al.* state that to reasonably use a full-fledged NLI, users need to be trained to use the system expediently [Androutsopoulos et al., 1995, Thompson et al., 2005]. Ginseng, in contrast, guides users through the query formulation and, thus, largely circumvents these problems and, in particular, the habitability problem. Pairing our habitability hypothesis with the results discussed in the lit-

Figure 4.17: The Ginseng user interface showing the instances of the class "lake" identified by red circles with an "I" on the right side of the user interface.

erature [Androutsopoulos et al., 1995, Bell and Rowe, 1992, Dekleva, 1994] suggests that Ginseng would be easier to use for the casual and occasional end-user than a full NLI approach. Consequently, as structuration theory [Giddens, 1984, Orlikowski, 1992] suggests, Ginseng aims at imposing a reasonable amount of structure on the query language in order to enable peoples' querying performance. By lying at the middle of the Formality Continuum, the interface provides a guide to the user without overly limiting his/her expressibility.

4.3.2 Technical Overview

From an architectural point of view Ginseng has four parts: a grammar compiler, a partially dynamically generated multi-level grammar, an incremental parser, and an ontology-access layer (cf. Figure 4.19). The first three parts were developed as part of the Ginseng project; for the ontology-access layer we use Jena's SPARQL engine ARQ.

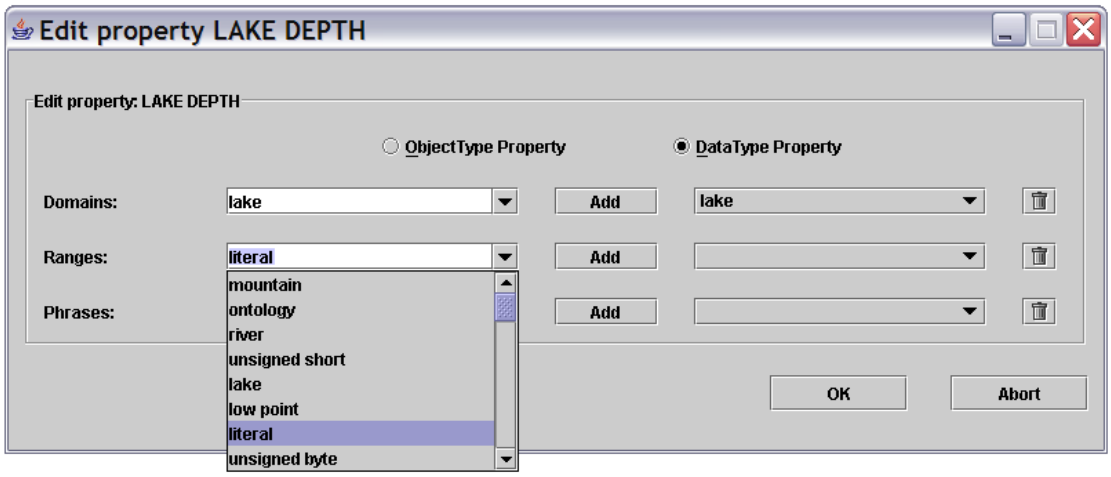


Figure 4.18: Ginseng's property editing window when clicking on the property "hasDepth" in the graph representation of the user interface. The window allows the specification of the property's type (datatype or object), its domain as well as range, and synonyms of the property's name (phrases).

When starting Ginseng, all knowledge bases in a predefined search path are loaded. For each knowledge base, the *grammar compiler* generates the necessary dynamic grammar rules in order to extend the static part of the grammar, which contains the ontology-independent rules specifying general sentence structures. The multi-level *grammar* is then used by the incremental parser in two ways:

1. First, it specifies the complete set of parsable questions, which can be used to provide the user with alternatives during entry as described above and to prevent incorrect entries.

2. Second, it also explains how to construct the SPARQL statements from the queries entered by the user. Thus, the complete parse tree of an entered question can be used to generate the resulting SPARQL query, which will then be executed with Jena's SPARQL engine ARQ.

The *incremental parser* maintains an in-memory structure representing all possible parse paths of the currently entered sequence of the characters. This has various benefits. First, it allows the parser to generate a set of possible continuations (i.e., possible next character sequences by expanding all existing parse paths, which are displayed by Ginseng's pop-up). One parse path might generate multiple options when the parser expands a non-terminal being specified in more than one place in the grammar. Second, the parser can compare every character entered against the possible entries, thus providing immediate feedback when the user attempts to enter an non-interpretable

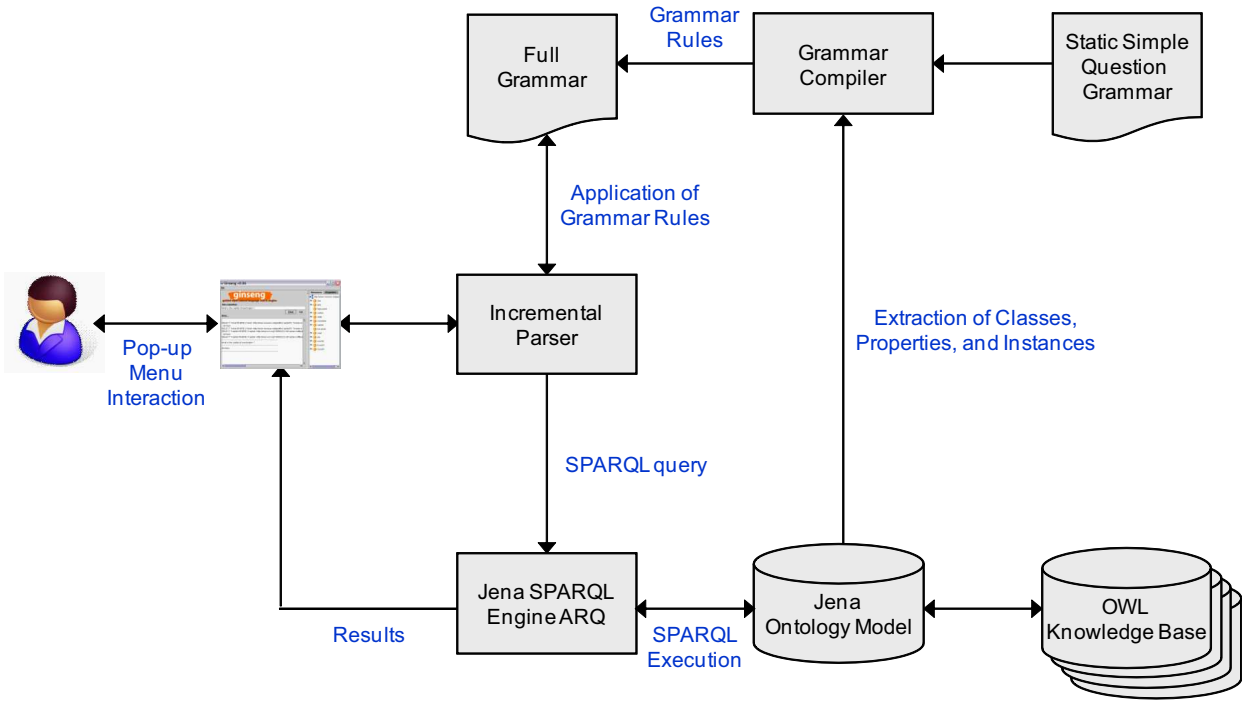


Figure 4.19: The Ginseng architecture featuring a grammar compiler, a multi-level grammar consisting of a static part that is extended by a dynamic part generated by the grammar compiler and both parts build the full grammar, an incremental parser, and the ontology access layer Jena.

(i.e., non-parsable) sentence/character to mitigate the habitability problem. Third, when the user has finished entering the sentence, the parser can immediately provide the set of acceptable parse paths. When querying, a simple transformation relying on the above mentioned query construction grammar can translate the parse paths to SPARQL queries, avoiding the lengthy semantic interpretation (and possible delays in answering the query) of the sentence that is usual in NLIs [Androutsopoulos et al., 1995] [Turtle, 1994].

Note that, since query fragments can be parsed in multiple different ways, the grammar is not deterministic, thus there might be multiple parse paths—many of which could be ambiguous. Consequently, the parser needs to keep track of all possible parse trees as alternative "hypotheses." When completing the query entry, the number of possible parse trees is usually reduced. Nevertheless, it can occur that more than one possible parse tree exists at the end of the query entry, which causes Ginseng to generate a collection of SPARQL queries. The collection of SPARQL queries is then passed to Jena, and the sorted set of the union of all answer sets, with duplicates removed, is returned to the user.

4.3.3 The Ginseng Grammar

The Ginseng grammar describes the parse rules of the English queries entered by the user as well as the query composition elements of the corresponding SPARQL queries. Consider the grammar excerpt in Figure 4.20 as an example.

```
(1)  <START>     ::=  <OQ> ?
                       | SELECT <<OQ>>
                       | WHERE { <<OQ>> }

(2a) <OQ>        ::=  which <subject> <verb>
                       | <<subject>>
                       | <<subject:1>> <<verb>>

(2b) <OQ>        ::=  what <subject> <verb>
                       | <<subject>>
                       | <<subject:1>> ) ( <<subject:1>> <<verb>>

(3)  <subject>   ::=  state
                       | ?state
                       | <rdf:type> <geo:state> ( type=[<geo:state>] )

(4)  <verb>      ::=  borders <object>
                       | -
                       | <geo:borders> <<object>> ( domain=[<geo:state>],
                             range=[<geo:state>] )

(5)  <object>    ::=  new york city
                       | ?newyorkcity
                       | <geo:newYorkCity> ( type=[<geo:city>, <geo:capital>] )

(6)  <object>    ::=  mississippi
                       | ?mississippi
                       | <geo:mississippi> ( type=[<geo:river>] )

(7)  <object>    ::=  mississippi
                       | ?mississippi
                       | <geo:mississippi> ( type=[<geo:state>] )
```

Figure 4.20: A grammar excerpt in Backus-Naur-Form notation considered as an example for the explanation of the functionality of the Ginseng grammar. <OQ> stands for *object query*.

The grammar's representation mostly follows the Backus-Naur-Form notation: nonterminal symbols use uppercase characters (e.g., <OQ>), whereas terminal symbols such as `state` that can be displayed to the user in a pop-up use lowercase characters. Grammar elements after the pipe "|" symbol denote type restrictions. Note that, to keep things understandable, we have simplified the example rules by removing full URIs, but we preserve angle brackets, which are usually only used with resources and their full URIs. While parsing a query entered by the user, the incremental parser searches recursively for possible matches to the symbols on the left side of the rules, and replaces them with the symbols on the right side of a conformable rule. The parse is completed when no

non-terminal symbols remain. By keeping every replacement step during the parsing process, a parse tree of an entered query is successively built.

The pipe symbol "|" is used to separate the parse elements of each rule from the associated query construction elements. Each line of a rule's right side, which is separated by the pipe symbol from the other lines, is consequently replaced by the same line of another conformable rule during a parsing process. To state a logical "OR" in the grammar despite the already used "|" symbol, one simply repeats the rule's head, which causes the Ginseng parser to interpret the two rules as alternatives. Rules (2a) and (2b) are such alternatives.

We first focus our attention on the parts of the grammar before the first "|" (i.e., the first line of each rule). Every sentence begins with the `<START>` symbol. To replace the `<START>` symbol, this simple grammar offers the non-terminal symbol `<OQ>` (standing for *object query*) followed by the terminal symbol "?" (rule (1)). It then searches for possible matches for `<OQ>`, finds them in rules (2a) and (2b), and then proposes to start the query with the terminal symbols "which" or "what," with each being followed by a `<subject>` and a `<verb>`. The terminal symbols "which" and "what" are displayed to the user in a pop-up menu as possible beginnings of an entry. If the user enters "what," then the parser can bind `<OQ>` to rule (2b) and discard (2a) as a possible parse hypothesis.

Next, the parser tries to match the non-terminal symbol `<subject>`, for which this grammar only offers rule (3). Hence, `<subject>` is replaced with the terminal symbol "state," which is followed by the non-terminal `<verb>` still remaining from rule (2a). `<verb>` is matched by rule (4) and replaced by the terminal symbol "borders", resulting in the sentence fragment "`What state borders <object> ?`" (the question mark was introduced in rule (1)). For the resolution of the last non-terminal symbol, `<object>`, rules (5), (6), or (7) could be applied, leading to the three alternatives: "new york city," "mississippi," and "mississippi." To offer only valid alternatives as possible entries to the user, Ginseng now makes use of the grammar rules' parts after the second "|," i.e., the third line of each rule.

When entering the word "borders", and thus prompting the choice of rule (4), the parser also received the information that the entered `<verb>` requires the following `<object>` to be of type `<geo:state>`. According to the RDF subject–property–object structure, the type information is defined by `range = [ <geo:state> ]` in the third line of rule (4) and constrains the choices of the object following the verb/property. Only the `<object>` in rule (7) fulfills the constraint of possessing the type `<geo:state>`, therefore excluding rule (5), as "new york city" is a `<geo:city>` and a `<geo:capital>`, as well as rule (6), which treats the river "mississippi" rather than the state. As a consequence the remaining rule (7) is used.

Additionally, the domain specification `domain = [ <geo:state> ]` in the third line of rule (4) also demands a specific type from the subject of the verb "borders," namely

to be of type <geo:state> too. When entering the word "state," and thus binding <subject> to rule (3), the parser had already received the information that the entered <subject> was of RDF type <geo:state>, therefore satisfying the required domain constraints of the verb in rule (4).

The last symbol remaining is the question mark from rule (1). If the user enters or chooses it, the query is complete: "What state borders Mississippi?" As there are no symbols left, and all type, domain, and range requirements are satisfied, the resulting valid parse tree is as follows:

```
<START>
 |- <OQ>
 |    |- "what"
 |    |- <subject>
 |    |    \- "state"
 |    \- <verb>
 |         |- "borders"
 |         \- <object>
 |              \- "mississippi"
 \- "?"
```

Behind the first "|" symbol, i.e., the second line of each rule, each grammar rule contains the information needed to assemble the SELECT statement of the SPARQL query, which it does by following the parse tree in a bottom-up fashion. In these second lines, the grammar's non-terminals are shown in double arrows (e.g., <<subject>>) to distinguish them from the abbreviated URIs, which also hold arrows in their notation. Taking our example query and the SPARQL assembly instructions after the first "|" of rule (1), replacing the non-terminal <<OQ>> with <<subject>> of rule (2b), and then with ?state according to rule (3), the query results in the "SELECT" statement

```
SELECT ?state .
```

The third part of each grammar rule (the third line of the rules) is used to construct the "WHERE" statements of the SPARQL query. It, furthermore, includes type, domain, and range information in order to constrain the subjects and objects of properties such that only valid queries can be entered into a loaded knowledge base (as we have encountered above). The third line of each grammar rule can have back references to the second part/line of the rule: In rule (2a), for example, <<subject:1>> refers to the symbol <<subject>> in the second query line, ensuring that both are bound to the same symbol. The number "1" facilitates the numbering and, therefore, the differentia-

tion of variables if the same terminal symbol is used more than once in the same query; the recursively designed grammar allows the reuse of the same non-terminal symbols at different positions in a question. Assembling the information in each third part/line by incrementally replacing the symbols of one rule with the symbols of the next matching rule (including brackets) until no non-terminals are left results in the statements:

```
WHERE {
   ?state <rdf:type> <geo:state> (type=[<geo:state>])
   ?state
      <geo:borders> (domain=[<geo:state>],range=[<geo:state>])
      <geo:mississippi> (type=[<geo:state>])
}
```

In a post-processing step, this notation is converted into correct SPARQL syntax, a "FILTER" statement is inserted, the additional but no longer necessary type, domain, and range information in parentheses removed, and the complete, coherent SPARQL query is passed to the Jena ARQ for execution. The resulting SPARQL query (with abbreviated URIs) for the question "What state borders Mississippi?" is:

```
SELECT ?state
WHERE {
   ?state <rdf:type> <geo:state> .
   ?state <geo:borders> ?instance .
      FILTER REGEX (?instance, "mississippi") .
}
```

After executing this query with Jena against the OWL ontology specified by the URI of the namespace "`geo`," we acquire the correct answer:

```
Alabama, Arkansas, Louisiana, Tennessee.
```

When matching rules, the grammar also exploits concept hierarchies. Consider a property that connects a state to the cities that lie within this state (e.g., `<geo:hasCity>`). The property's range constrains the possible objects of type "city." Nevertheless, as "capital" is a sub-class of the class "city," Ginseng would offer both concepts as possible object entries for the property to the user. In contrast, sub-property relationships are not exploited, since sub-properties can be named very differently than their super-properties—in most cases leading to user confusion. Seeking not to confuse the casual user not familiar with RDF or OWL structures, we decided not to provide ontology information, such

as that of classes, within the answer set.

4.3.4 The Grammar Compiler

When loading an ontology, Ginseng generates a dynamic grammar rule for every class, property, and instance. These dynamic rules enable the display of the labels used in the ontology in pop-up boxes. While the *static grammar rules* (rules (1), (2a), and (2b) of the example grammar excerpt in Figure 4.20) provide the basic sentence structures, the *dynamic rules* (rules (3) to (7) in Figure 4.20) are generated from a knowledge base that allows certain non-terminal symbols of the static rules to be "filled" with terminal symbols (i.e., the IDs of resources and labels of instances) that are extracted from the knowledge base including its ontology model.

The *static grammar rules* provide the basic phrase and sentence structures for questions. They are completely independent of domains and knowledge bases. They handle general questions, such as the example from above, "`What` state borders Mississippi`?`" (static grammar terminals in teletype font), and also other types of queries, for example closed questions ("`Is there` a city `that` is the highest point of a state`?`"), which typically result in an answer of "yes" or "no"), or questions resulting in numerical answers (e.g., "`How many` rivers run through Georgia`?`"). Furthermore, it provides sentence construction rules for the conjunction or disjunction of two phrases (or sentence parts). The static grammar consists of about 120 mostly empirically constructed domain-independent rules, a number that could be easily extended. Nevertheless, given the findings in the literature [Chakrabarti, 2004, Dittenbach et al., 2003] [Linckels and Meinel, 2006, Malhotra, 1975, Spink et al., 2001], we believe that the grammar does not need to be complete, as people tend to use a limited language when interacting with a system interface.

To provide an impression of the static rules other than (1), (2a), and (2b) in Figure 4.20, consider another grammar excerpt as displayed in the following. Here,

```
<SPO_OBJ> ::= what is [<det>] <NC>
              | ?class
              | ?class <rdfs:subClassOf> <<NC>>
```

parses questions such as "What is a city?", and

```
<SPO_OBJ> ::= what is [<det>] <NI>
              | ?instance
              | <<NI>> <rdfs:type> ?instance
```

parses questions such as "What is the McKinley?", generating answers such as "mountain," "thing," and "resource." Both rules use the optional non-terminal <det>, which is defined as follows:

```
<det> ::= a
<det> ::= the
```

Furthermore, <NI> stands for noun-instance and <NC> for noun clause. Their defining rules are part of the dynamic grammar.

The *dynamic grammar rules* are generated by the loaded OWL knowledge bases (rules (3) to (7) in Figure 4.20). The grammar compiler essentially parses a knowledge base and generates a rule for each class, instance, object property, and datatype property. To illustrate the dynamic rule generation, we will show the translations of an OWL class, instance and property into their corresponding generated rules.

Consider the following OWL class definition in a file where the prefix "ginseng" is assigned to the namespace URI {http://www.ifi.uzh.ch/ddis/ont/nli/geo.owl#}:

```
<owl:Class rdf:ID="State">
   <ginseng:phrase rdf:value="states"/>
</owl:Class>
```

Its transcription generates two Ginseng rules for noun clauses; one for the actual class definition and one for the "ginseng:phrase" tag, which facilitates the use of noun plurals. Since the noun starts with a consonant, the resulting rules describe the non-terminal <NCc> as follows (rather than <NCv> for nouns starting with a vowel):

```
<NCc> ::= state
        | ?state
        | <geo:state>

<NCc> ::= states
        | ?state
        | <geo:state>
```

The distinction between nouns starting with a vowel or a consonant is necessary, as a preceding determiner (such as "a" or "an") might constrain a subsequent noun to the state of being either consonant- or vowel-initial. Thus, we handle determiner-noun

agreement (e.g., "a class" vs. "an instance"). Analogously, subject-predicate agreement is controlled (e.g., "Which state is ..." vs. "Which states are ..."), as illustrated in the object property example "`borders`" below.

An OWL instance such as

```
<State rdf:ID="mississippi">
   <rdfs:label xml:lang="en">mississippi</rdfs:label>
   <borders rdf:resource="#arkansas" />
   ...
</State>
```

gets straightforwardly transformed into:

```
<NI> ::= mississippi
         | ?mississippi
         | <geo:mississippi>
```

OWL object properties typically denote verbs (or predicates) and are typically followed by an object. Consequently, the verb rules usually contain a non-terminal symbol referring to an object. Thus,

```
<owl:ObjectProperty rdf:ID="borders">
   <rdf:type rdf:resource="owl:SymmetricProperty" />
   <rdfs:domain rdf:resource="#State" />
   <rdfs:range rdf:resource="#State" />
   <ginseng:phrase rdf:value="border"/>
   <ginseng:phrase rdf:value="surrounds"/>
   <ginseng:phrase rdf:value="surround"/>
   <ginseng:phrase rdf:value="adjoin"/>
   <ginseng:phrase rdf:value="adjoins"/>
   ...
</owl:ObjectProperty>
```

is translated to

```
<verb> ::= borders <object>
           | -
           | <geo:borders> <<object>>
                (domain=[<geo:state>], range=[<geo:state>])
```

The translation of datatype properties is slightly more complicated. Datatype properties, e.g., <geo:hasPopulation>, <geo:hasLength>, and <geo:hasStateArea>, are usually treated as nouns in questions such as "What is the population of Boston?" or as adjectives as in "How big is Boston?". The verb "has," which is part of the property's ID name, is completely ignored, and only appears in questions such as "What population has Boston?". Our grammar compiler takes these differences into account and generates six grammar rules for each datatype property plus three rules to facilitate the alternative "which" to "what" at question beginnings. The translation of the OWL datatype property,

```
<owl:DatatypeProperty rdf:ID="hasPopulation">
   <rdfs:domain rdf:resource="#City"/>
   <rdfs:range rdf:resource="xml:float"/>
   <ginseng:ignore rdf:value="id text"/>
   <ginseng:phrase rdf:value="population"/>
   <ginseng:interrogative rdf:value="how big"/>
</owl:DatatypeProperty>
```

thus results in the nine grammar rules depicted in Figure 4.21 (<SQ> stands for *subject query*). Rules (8) and (10), together with some of the static rules, provide the necessary question parsing and SPARQL query composition information for the translation of the question "How big is Boston?" into the following simple SPARQL query:

```
SELECT ?howbig
WHERE {
   ?city <rdf:type> <geo:city> .
   ?city <geo:hasPopulation> ?howbig .
      FILTER REGEX (?city, "boston") .
}
```

After demonstrating the translation of OWL elements into grammar rules for use by Ginseng's parser, we must now explain the tags ginseng:phrase, ginseng:ignore, and ginseng:interrogative, which we introduced in the definitions of object property <geo:borders> and datatype property <geo:hasPopulation> above.

As seen above, plural forms of nouns and verbs can easily be included into an ontology by inserting the corresponding forms via the tag ginseng:phrase. The additional forms annotated with this tag can also be displayed to the user in the pop-up menus,

```
(8)  <SQ>                 ::=  how big <verb_population> of <Subject>
                                |?howbig
                                |<<Subject>> <<verb_population>> ?howbig

(9a)  <SQ>                ::=  what <verb_population> of <Subject>
                                |?what
                                |<<Subject>> <verb_population> ?what

(9b)  <SQ>                ::=  which <verb_population> of <Subject>
                                |?what
                                |<<Subject>> <verb_population> ?what

(10) <verb_population>    ::=  is [<det>] population
                                |-
                                |<geo:hasPopulation>
                                    ( domain=[<geo:city>], range=[<xml:float>] )

(11a) <SQ>                ::=  what <noun_population> has <Subject>
                                |?what
                                |<<Subject>> <<noun_population>> ?what

(11b) <SQ>                ::=  which <noun_population> has <Subject>
                                |?what
                                |<<Subject>> <<noun_population>> ?what

(12a) <SQ>                ::=  what <noun_population> does <Subject> have
                                |?what
                                |<<Subject>> <<noun_population>> ?what

(12b) <SQ>                ::=  which <noun_population> does <Subject> have
                                |?what
                                |<<Subject>> <<noun_population>> ?what

(13) <noun_population>    ::=  population
                                |-
                                |<geo:hasPopulation>
                                    ( domain=[<geo:city>], range=[<xml:float>] )
```

Figure 4.21: The grammar rules dynamically generated by Ginseng's grammar compiler for the datatype property `<geo:hasPopulation>` when loading a knowledge base with geographic information.`<SQ>` stands for *subject query*.

thus increasing the query vocabulary.

The Ginseng tag `ginseng:ignore` suppresses the literal construction of a dynamic rule with the terminal symbol of a property's ID. Consider once more the datatype property `<geo:hasPopulation>`, for which the grammar compiler, due to the use of different linguistic expressions in questions asking for the population of a city, did not create a rule with the wording "has population." This suppression was because of the `ginseng:ignore` tag. The tag is especially useful when an OWL property ID does

not correspond to an English expression that can be used verbatim in a query (e.g., `rdf:ID="hiPoint"` actually meaning "is the highest point of").

Another possibility for influencing the vocabulary produced when automatically generating the dynamic grammar rules is the `ginseng:interrogative` tag. When building an ontology, the creator can use it to include question words for classes and properties, for example "how many," "how big," "how high," "how long," etc. in an annotated form. For each interrogative value, Ginseng's grammar compiler will also generate a corresponding grammar rule that must be compliant with the other rules associated with the resource. In this way, grammar rule (8) in Figure 4.21, which facilitated the translation of "How big is Boston?" into SPARQL statements, was created.

Furthermore, Ginseng permits any synonyms of the IDs used in the ontology model to be included by annotating the ontology with additional tags from the `ginseng` namespace (as already encountered above with the properties `<geo:border>` and `<geo:hasPopulation>`). Consequently, Ginseng generates a dynamic grammar rule for each synonym. The following class "water area" would, therefore, result in six corresponding grammar rules:

```
<owl:Class rdf:ID="waterArea">
   <ginseng:phrase rdf:value="water areas"/>
   <ginseng:phrase rdf:value="body of water"/>
   <ginseng:phrase rdf:value="bodies of water"/>
   <ginseng:phrase rdf:value="aquatic area"/>
   <ginseng:phrase rdf:value="aquatic areas"/>
</owl:Class>
```

While all annotations with Ginseng tags are not necessary for Ginseng to run correctly, they do extend its vocabulary and increase its overall usability. Additionally, they reduce the limitation of the approach, to some extent, depending on the choice of vocabulary when the ontology was built (this is also the case with NLP-Reduce and Querix). In fact, the more meaningful the IDs and the labels of a knowledge base are chosen, the wider and more useful the vocabulary provided by Ginseng is. Annotation consists either of simply adding new properties from the `ginseng` namespace to the ontology model or using Ginseng's editing window, which allows the specification of synonyms in a familiar, interactive way. The effort is manageable, since, if this process is desired, only the ontology model requires annotating. The mechanism allows knowledge base-specific adaptation, while Ginseng's overall design is kept completely portable in order to circumvent the adaptivity barrier.

4.4 Semantic Crystal

Our last interface, Semantic Crystal, has the most formal and most restrictive query language of the four systems [Sprenger, 2006, Kaufmann and Bernstein, 2007]. In order to compare the other NLIs with a more formal approach (though it is not full formal logic), and keeping in mind that casual end-users are better at understanding graphical query interfaces than formal query languages [Spoerri, 1993], we constructed the graphical query tool Semantic Crystal, which embeds the fully formal query language SPARQL. However, it is more intuitive for end-users due to its graphical display of the query language.

One approach we examined when sketching the Semantic Crystal system was the graphical graph query language *QGraph* by Blau *et al.*, which provides a rich graphical query language for graph structures [Blau et al., 2002]. The problem with approaches such as QGraph is, however, that they essentially translate the full complexity of a formal query language to a graphical layout, making the graphical layout of the query easier to understand than the textual, but still probably leaving it too complicated for the casual or occasional user (see an example QGraph query in Figure 4.22). We, therefore, concluded that a QGraph-like language needs to be simplified by using a more intuitive approach, through which casual end-users can construct queries without permanent support.

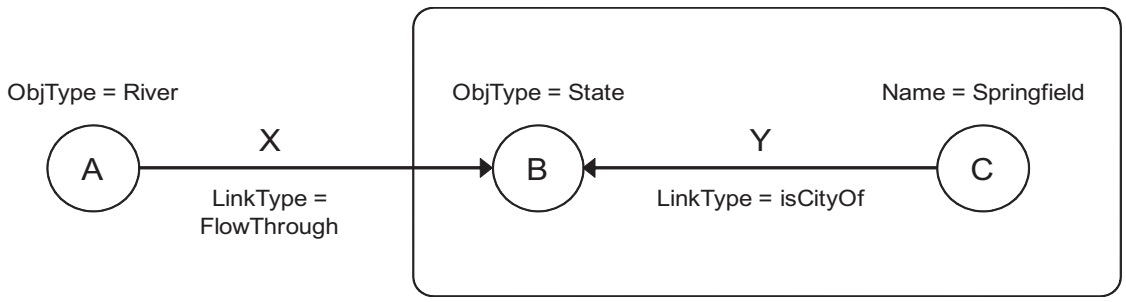


Figure 4.22: QGraph query that finds subgraphs containing rivers that flow through states of which Springfield is a city.

Addressing this issue, we drew inspiration from *InfoCrystal* by Spoerri [Spoerri, 1993], a graphically-based query tool for Boolean and vector space information retrieval. The name *Semantic Crystal* is, in fact, an homage to Spoerri's InfoCrystal. He demonstrated that the graphical InfoCrystal concept was easier to understand than traditional Boolean statements. Combining the QGraph approach with InfoCrystal-like ideas and the typical RDF triple graph representation for condition formulation, we developed a version of Semantic Crystal that features a graphical query language for SPARQL and takes advantage of the ontology in order to simplify the visual language. It accomplishes this

by displaying a query as shown in Figure 4.23, which is Semantic Crystal's graphical representation of the question depicted in QGraph in Figure 4.22.

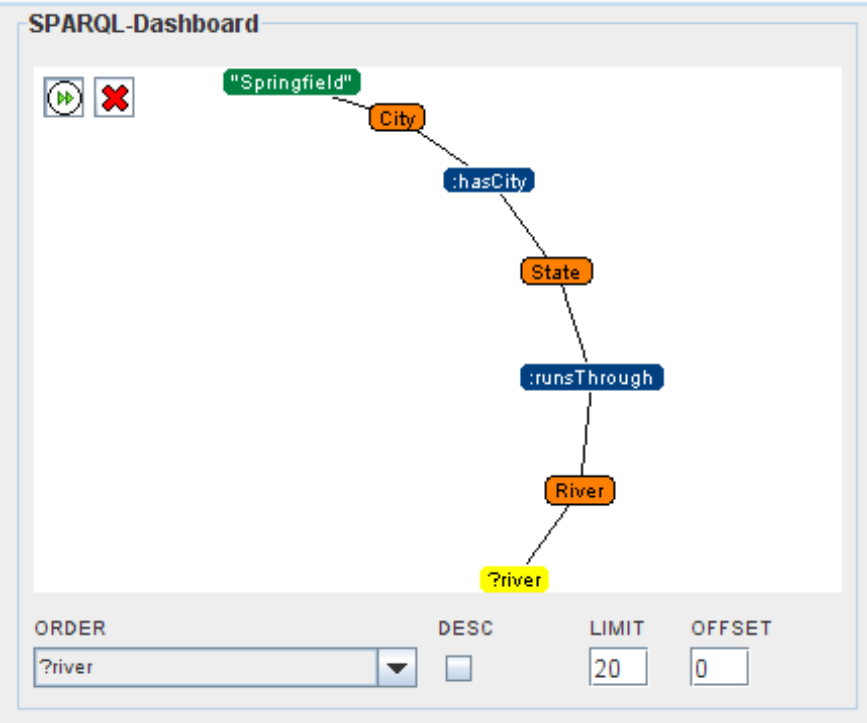


Figure 4.23: The Semantic Crystal SPARQL dashboard showing the graphical representation of the query "Give me the rivers that flow through the states that have a city named 'Springfield'."

Semantic Crystal is a domain-independent interface that can be used for querying any OWL-based knowledge base that is locally stored or linked somewhere on the Web. It also displays the ontology model, which users tend to find an advantageous feature. Queries are composed by clicking on elements in the graph and selecting elements from menus (see Figure 4.24). Once an element has been selected, the interface presents it on the SPARQL Dashboard on the upper right side of the user interface. The user may then continue assembling the query either on the dashboard or in the graph representation of the ontology model.

Semantic Crystal offers a plain opportunity for graphically constructing SPARQL queries. However, it is not an NLI, and it doubtlessly demands more abstraction skills or logic skills from its users. Though more challenging, the interface still addresses casual end-users—it is easier to master than a formal query language such as SPARQL. Some instructions and training are necessary if a novice user wants to use the interface, but we nonetheless believe that visualization can massively simplify query formulation, and were assured of this by our usability study, in which casual first-timers were able to com-

pose proper queries in Semantic Crystal by themselves after being given a brief written introduction to the system (see Chapter 6).

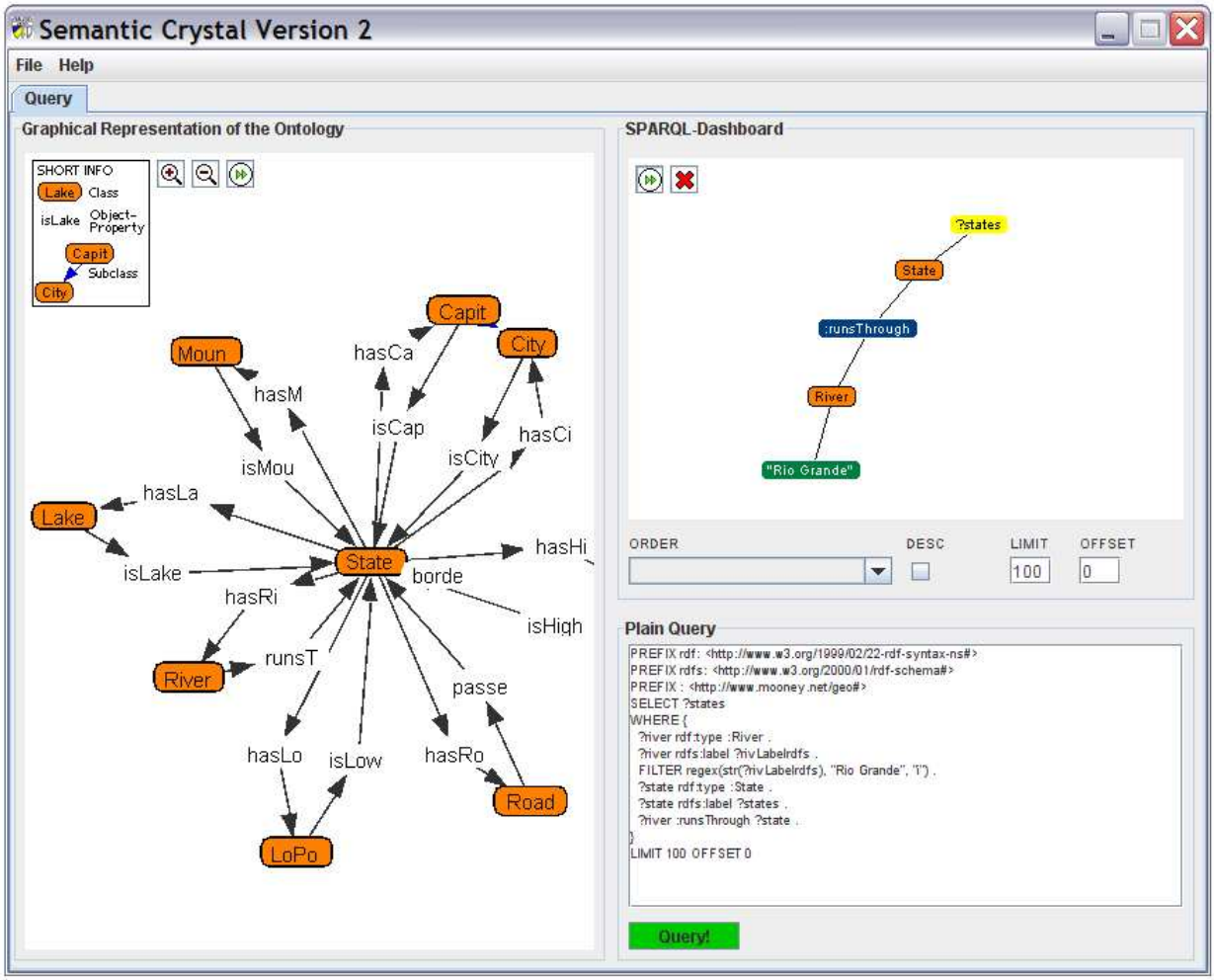


Figure 4.24: The Semantic Crystal user interface after composing a query searching for the states through which the Rio Grande flows.

As we have in the previous sections, we will now present the Semantic Crystal user interface by illustrating how users can compose queries with the search interface. Then we will describe the technical design of Semantic Crystal, which is notably simpler than those of the other three systems. This simple system design reflects that the burden of query construction and triple statement joining lies entirely with the user.

4.4.1 Querying in Semantic Crystal—The User Experience

Semantic Crystal provides users querying Semantic Web data with a graphically displayed, clickable, formal query language for SPARQL. As such, users with no SPARQL or ontology-based knowledge whatsoever can search OWL knowledge bases stored either on the Web or locally and construct precise queries. The interface closely follows the basic RDF and OWL concepts by also incorporating their typical graph representation where classes are represented by nodes and properties by arcs linking the nodes.

When starting Semantic Crystal, the user can choose the knowledge base that should be loaded by clicking on "File" in the menu of the interface (cf. Figure 4.24). Knowledge bases can be stored locally or on the Web; the user simply enters a URL in order to load a knowledge base that is provided on the Web. Under "File," one can also reload a knowledge base or close result tabs. Clicking on "Help" opens an HTML page that explains how to use Semantic Crystal and how one can construct queries with the interface in a browser window. The help is provided in English and German.

The field labeled "Graphical Representation of the Ontology" on the left side of the interface displays the ontology model of a currently loaded knowledge base. All nodes are equally important, a root node is not required and cycles between nodes are allowed. A "short info" box on the upper left side provides a quick explanation of the concepts depicted in the graphical representation. The "plus" and "minus" icons can be used to zoom in and out the ontology graph. Clicking on the green arrowhead causes an animated rearrangement of the graph. Each class of an ontology is displayed as orange element with its ID displayed as the element's label. Classes are connected by the object properties defined in the ontology, which are illustrated as two arcs and the ID of a property between the two arcs. Most IDs are presented in an abbreviated form in order to better facilitate the arrangement of big graphs. Nevertheless, the full ID text of a resource pops up if the mouse cursor is moved over an element.

A query is composed by clicking on elements in the graph and selecting elements from menus. Once an element has been selected, the interface presents it on the SPARQL dashboard on the upper right side of the user interface. The user can then continue assembling the query either on the dashboard or in the graph representation of the ontology model. All queries are based on what we call the *TORC approach*. As such, each query consists of at least these four TORC elements:

- token,
- output,
- restriction,
- connection.

Tokens comprise the classes of the ontology model (represented by the orange elements in the graph). The output determines what the interface should return to the user as result. Hence, output elements specify the variables of the SPARQL "SELECT" statement. When executing a query, they are bound to values of datatype properties. Values of datatype properties can also be used as restrictions to compose "FILTER" statements. Finally, object properties are used to connect the tokens, i.e., the classes of an ontology. In Semantic Crystal, a query is complete when the four TORC elements are specified. We will now demonstrate the construction of a query in Semantic Crystal by working through the simple example query "Which states have a city named Springfield?".

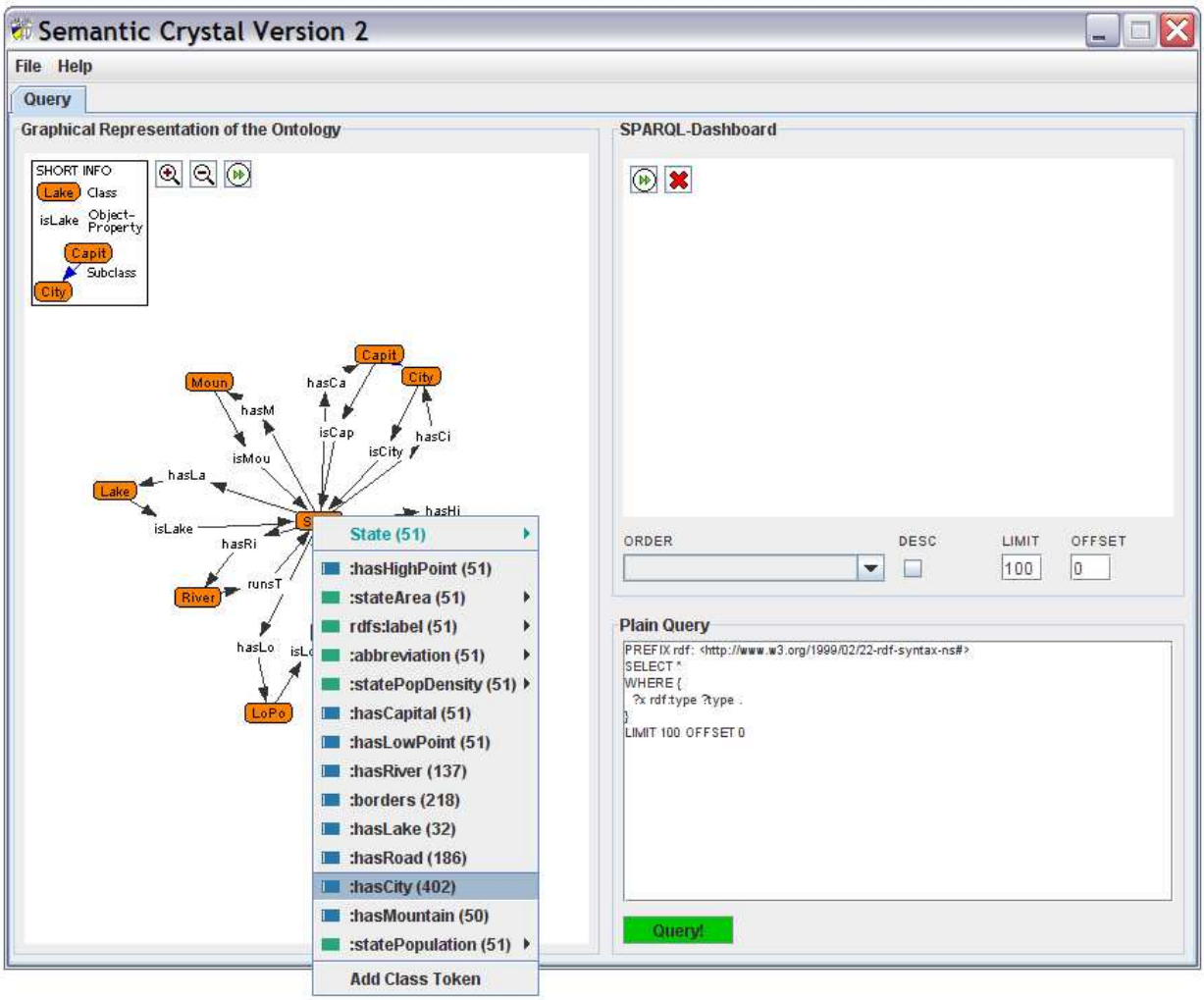


Figure 4.25: When clicking on a class (represented as the orange elements in the graph), the interface lists all properties of the class. Blue icons are used for object properties, green icons for datatype properties.

When clicking on a class such as "state," the interface lists all properties of the class as depicted in Figure 4.25, thus enabling the user to select one of them and, furthermore, to select only valid properties of that class. Following the Protégé convention [Stanford Medical Informatics, 2007], properties with blue icons are object properties, whereas properties with green icons are datatype properties. For each property, the number of instantiations is indicated in brackets. When selecting the property "hasCity," the property as well as the classes it relates are presented in the SPARQL dashboard on the upper right side of the user interface. Note that it does not make a difference whether the property "hasCity" of class "state" or the property "isCityOf" belonging to class "city" is chosen. Both properties connect the same classes vice versa leading to the same result as they are defined as inverse properties. The user can now continue with the construction in the SPARQL dashboard or in the graph representation.

Since the example query searches for a specific city, i.e., Springfield, this information is used as a restriction. Clicking on the element "city" opens all its properties in a menu list. In the case of datatype properties (the green properties), a user can specify if the property's value should be used as restriction or as output. We choose restriction in order to enter "Springfield" as one. By clicking on the newly added green element (representing restrictions), a text field opens where the restriction "Springfield" is entered (Figure 4.26).

The definition of restriction values for datatype properties in Semantic Crystal allows the use of regular expression patterns. They follow general pattern matching syntax rules and are included in the SPARQL query language. As such, the need for further SPARQL execution adaption is dispensable. Some of the patterns are listed in Table 4.4.

Pattern	Description	Examples
`^pattern`	matches at the start of a string	"pattern," "patternxyz"
`pattern$`	matches at the end of a string	"pattern," "xyzpattern"
`p{2}`	repeats the previous item exactly 2 times	"pp," "xppy"
`p{2,3}?`	repeats the previous item between 2 and 3 times	"pp," "ppp," "xppy"
`p*?`	repeats the previous item zero or more times	"xyz," "xp," "ppz"
`p+?`	repeats the previous item once or more	"p," "xp," "pppz"

Table 4.4: The regular expression patterns allowed in Semantic Crystal for specifying restrictions of datatype property values. They are included in the SPARQL query language.

Datatype values can also be restricted in arithmetic expressions, and queries such as "List the cities that have a population of over one million." can thus be answered. SPARQL supports arithmetic filtering in the "FILTER" statement, which could, for example, be defined as follows:

```
FILTER REGEX (?population > 1000000) .
```

Though regular expression patterns and arithmetic comparisons are very useful, they are not mandatory when defining a restriction in Semantic Crystal and, obviously, they are intended specifically for expert users.

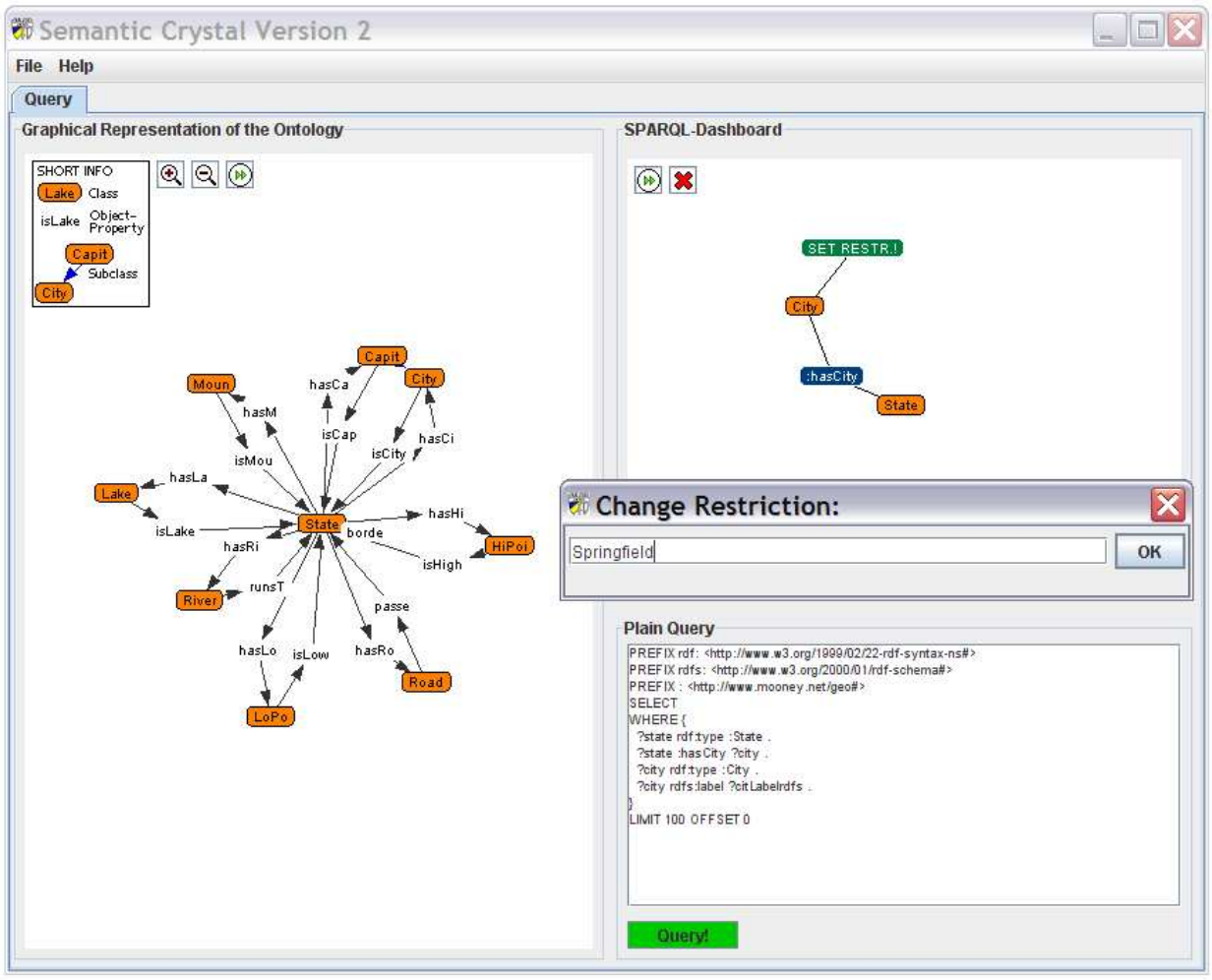


Figure 4.26: With datatype properties, a user can specify whether the property's value should be restricted by a string such as "Springfield".

The last TORC element to be specified for our example query is the output element. Only if an output is specified, the query can be executed. The output is specified by clicking on the class element "state" and selecting its label as output (see Figure 4.27). Additionally, a user can change the name of the output variable, if he or she so desires,

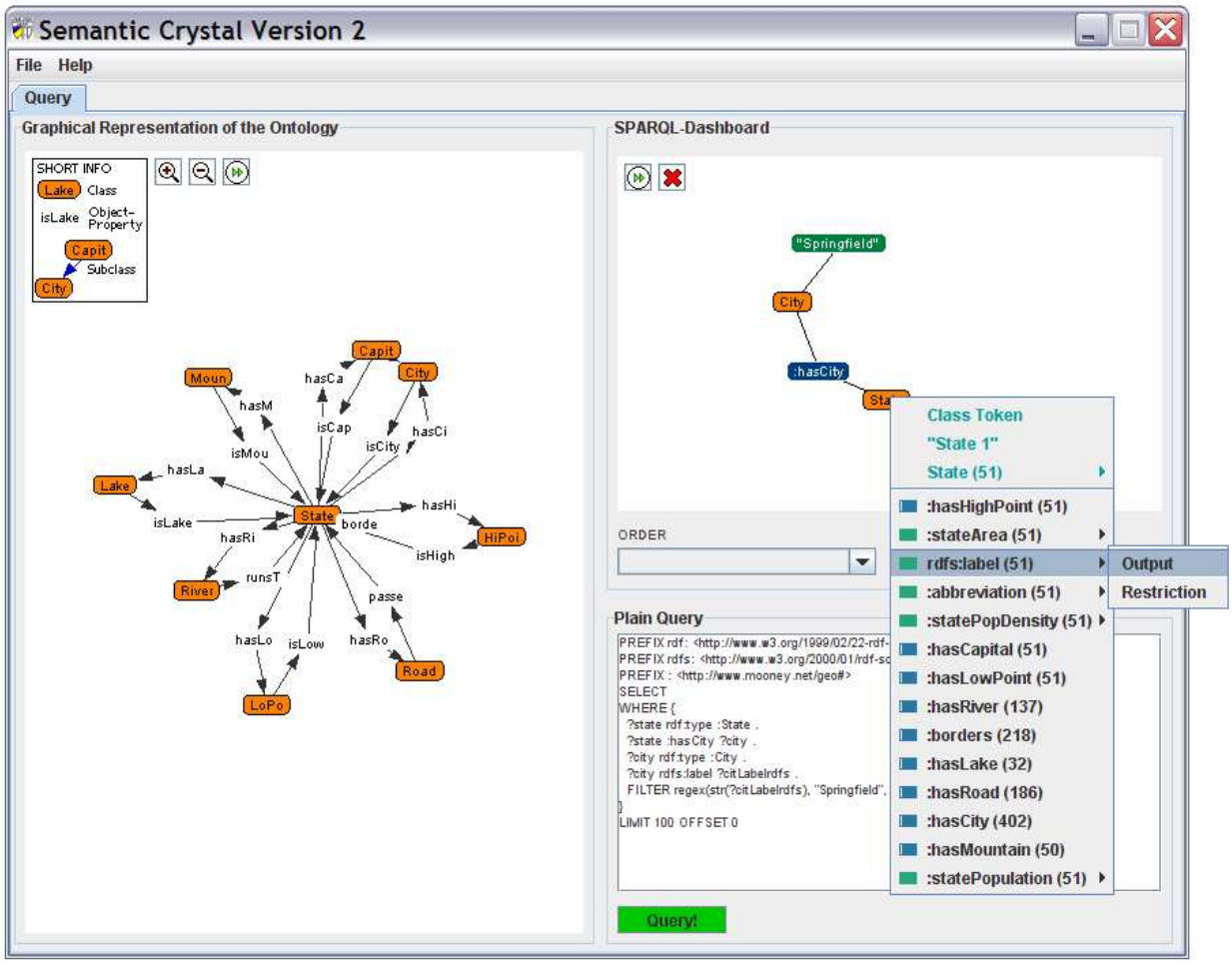


Figure 4.27: In the case of datatype properties (the properties with the green icons), a user can specify the output to be returned by Semantic Crystal.

by clicking on the yellow output element and entering the new variable identifier in a text entry field. By default, Semantic Crystal determines the names for variables on the basis of the IDs of classes.

After executing a complete TORC query by clicking on the green button labeled "Query!", the result is shown to the user in a new tab. The Jena SPARQL engine ARQ is again applied for query execution. On the dashboard in Fig. 4.28, we see a complete graphical representation of the example query "Give me the states that have a city named Springfield." and the result set retrieved by Semantic Crystal in Figure 4.29. By clicking on the "Query" tab, the user returns to the query composition view. At any stage of the query construction, single elements can be removed or the whole query deleted by clicking on the red "X" icon in the dashboard.

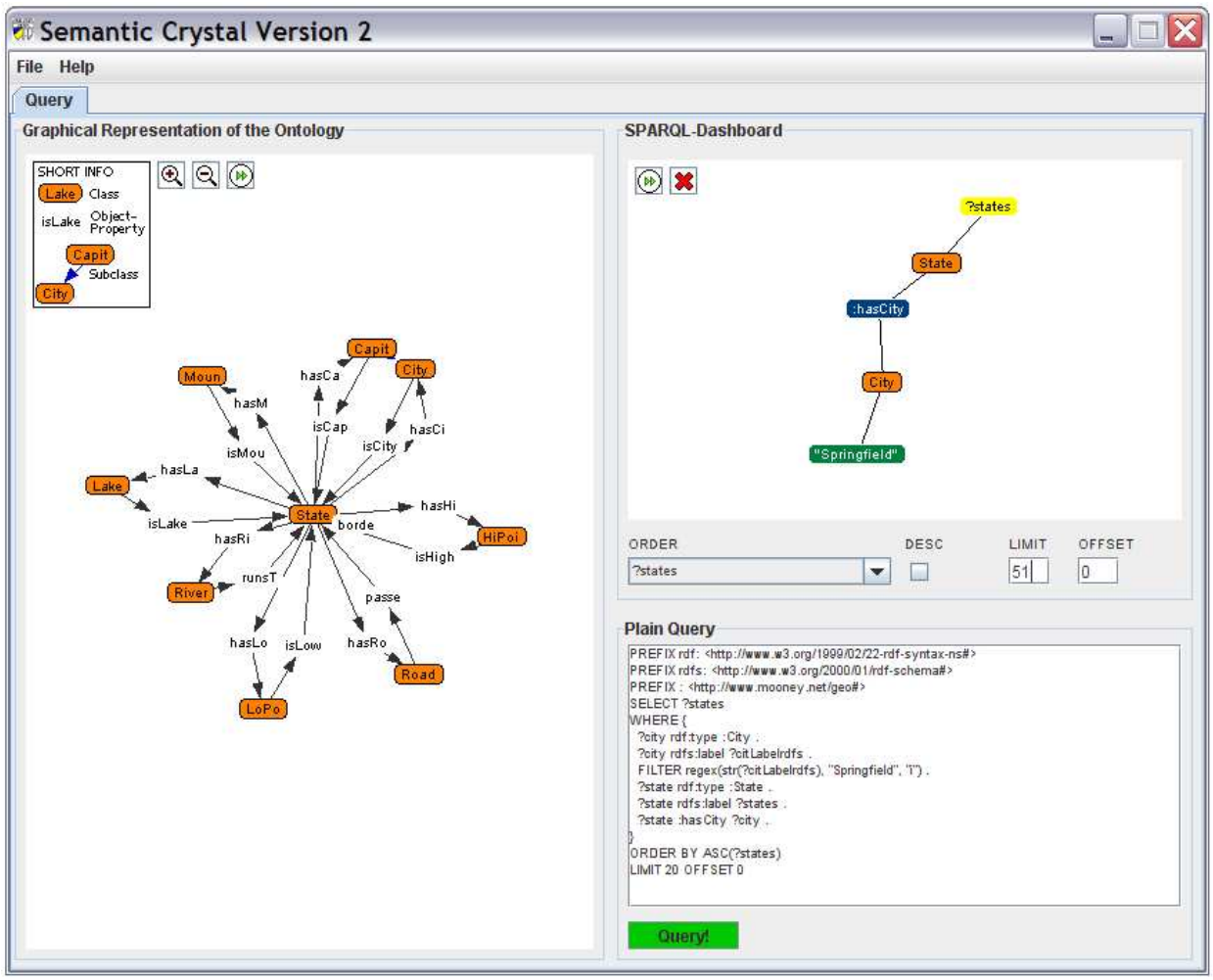


Figure 4.28: The Semantic Crystal interface showing a complete graphical representation of the query "Which states have a city named Springfield?" in the SPARQL dashboard on the upper right of the interface.

Before executing a query, a user can optionally specify solution sequence modifiers representing the applicable options in SPARQL by selecting on of the querys variables from the area immediately below the SPARQL dashboard in order to list the result set in either ascending or descending order. The choice is translated to the SPARQL clause `ORDER BY` with one of the optional order modifiers `ASC()` or `DESC()`. Using the "LIMIT" and "OFFSET" options in Semantic Crystal generates different subsets of query solutions. The corresponding SPARQL clause `LIMIT` places an upper bound on the number of solutions returned, while `OFFSET` causes the solutions to start after a specified number. Similar to regular expression patterns and arithmetic restrictions, the solution sequence modifiers are intended for experienced or expert users.

The interface incrementally generates textual SPARQL query statements for the current state of the graphically constructed query. The SPARQL statements are exhibited on the bottom of the right side of the user interface (cf. Figure 4.28). As such, the tool supports both an interactive layout and the editing of a query with immediate feedback via a display of the generated SPARQL statements. If a SPARQL expert user finds it faster or more convenient to formulate a query by directly specifying or modifying SPARQL statements, Semantic Crystal enables him/her to do so.

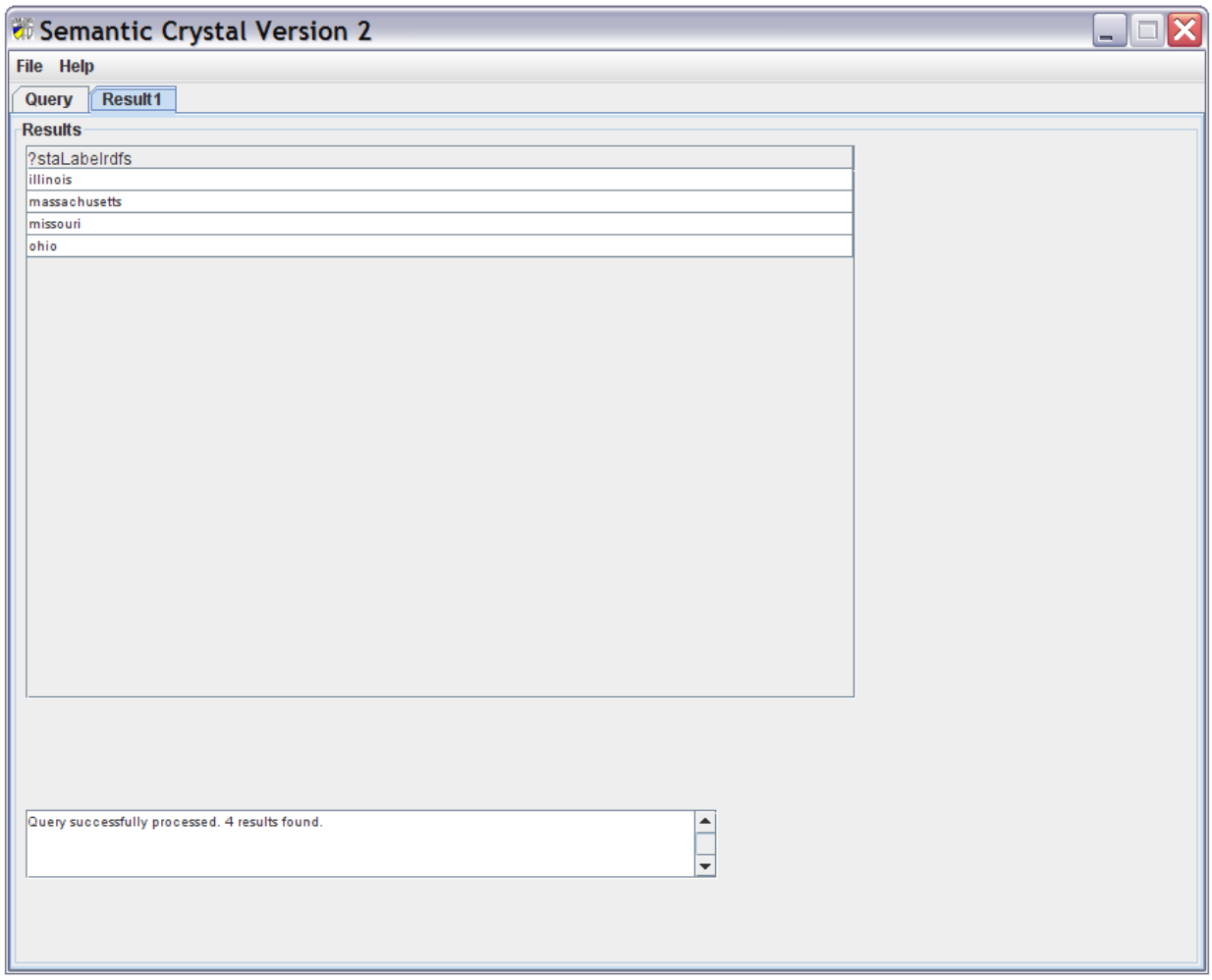


Figure 4.29: The result set to the query "Which states have a city named Springfield?" is returned in a new tab of the Semantic Crystal interface.

4.4.2 Technical Overview

The domain-independent interface can be used for querying any OWL-based knowledge base that is locally stored or linked on the Web. It is implemented in Java and displays the ontology model to the user as shown above, a feature that is advantageous for the and-user. In conformity with our adaptivity hypothesis, the system's architecture is rather simple, as shown in Figure 4.30, consisting of four major parts: a user interface, the Jena framework, the Semantic Crystal XML converter, and the Prefuse Visualization Toolkit.

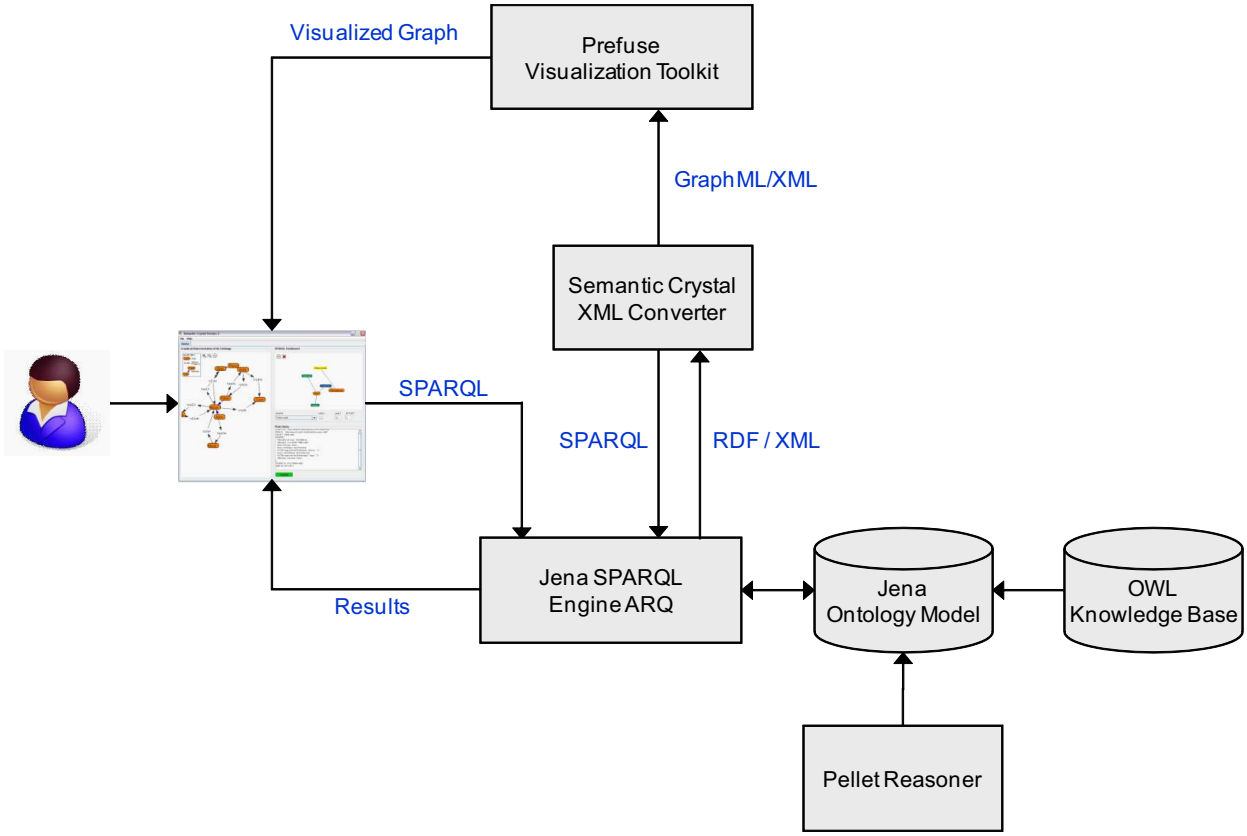


Figure 4.30: The simple architecture of Semantic Crystal featuring the Jena framework as ontology access layer and its ARQ as execution engine for SPARQL, the XML data converter that converts RDF/XML to GraphML/XML, and the Prefuse Visualization Toolkit which displays the ontology graph in XML format as visualized graph in the user interface.

The *user interface* provides a graphical query interface that embeds the formal query language SPARQL. It displays an ontology graph to the user and allows query construction with common interaction paradigms, i.e., clicking on elements and entering textual restrictions. It also displays the generated SPARQL statements to the user and returns the results after executing the SPARQL query in a new tab.

The *Jena framework* is again used as ontology-access layer. It first builds a Jena Model from an OWL knowledge base when loading it into Semantic Crystal. The Pellet reasoner provides inferred triples from given triples such as subclass relationships. A knowledge base that is fetched from the Web by declaring its URL is first stored locally, and, as such, there is no difference during query construction and execution between local or Web-based knowledge bases.

The *Semantic Crystal XML converter* transforms the Jena ontology model into a particular XML-based format such that the visualization tool is able to graphically represent the ontology model in the user interface. By firing SPARQL queries, the converter retrieves all classes, object and datatype properties and transforms their RDF/XML definitions into *GraphML* notation [Team, 2007]. GraphML is a file format for graphs whose syntax is based on XML. It is, therefore, ideally suited as language to graphically visualize RDF/XML triples. Moreover, it is fully compliant with the visualization library of Semantic Crystal.

The *Prefuse Visualization Toolkit* is used as graph library to visualize the OWL ontology models in Semantic Crystal [Heer, 2007]. It is an open-source Java library under GNU Free Documentation License[3]. Prefuse allows the physical animation of a graph where nodes behave like magnets, responding to each other by magnetic rejection and the arcs between the nodes are stretched and compressed like strings. Both the level of rejection and the elastic force of the springs can be specified in Prefuse; in Semantic Crystal, we decided for a calm animation. As Prefuse is based on XML syntax, it can easily be applied to the graphical display of ontology graphs.

When developing Semantic Crystal, we also evaluated the freely available GrOWL visualization and editing tool by Krivov [Krivov, 2007], which provides a graphic representation for OWL and DL ontologies. However, we concluded that GrOWL overloads users with information as it presents not only classes, subclasses, and properties but also all OWL property characteristics as well as instances. We think that showing all this information is too confusing. In our interface, datatype properties are only shown in a menu list when clicking on a class. The overall goal was a graphical layout close to RDF and ontology graph principles, where classes are the nodes and object properties the arcs between the nodes, but not too disincentive for casual end-users.

Just like every search system that relies on knowledge extracted from a knowledge base in order to employ the information in the query construction and translation process, Semantic Crystal depends highly on the vocabulary and quality of a knowledge base. Consider, for example, a graph in Semantic Crystal that only depicts some cryptic numbers as class and property names because the creator of the ontology used numbers as IDs for the resources. We are aware that such an ontology model, though graphi-

[3] `http://www.gnu.org/copyleft/fdl.html`

cally visualized, would be absolutely useless for casual end-users. Fortunately, ontology creators are mostly motivated on their own to choose meaningful labels for resources.

4.4.3 The XML Converter

In the last section of this chapter, we will describe how the XML converter of Semantic Crystal transforms RDF/XML triples into GraphML definitions, therefore, facilitating the graphical display of the ontology model in the user interface.

The XML converter's task is to retrieve all ontological data that should be displayed in the graphical representation of the Semantic Crystal interface from the Jena Model, and to convert the RDF data into the data format that is compliant with the Prefuse visualization library. Ontology data access is carried out through SPARQL queries. The converter first searches for all classes in the model whose RDF/XML notation is converted into XML format for node elements in GraphML. Next, a SPARQL query retrieves all object properties and transforms their statements into GraphML edge elements. The converter also retrieves subclass relations in order to display them in the graphical representation with blue arcs. Since Semantic Crystal runs into problems when an ontology model is not complete or erroneous, an error message is issued for these cases.

Datatype properties do not appear in the graphical representation of the ontology model, but in the submenus popping up when clicking on a class in the graph. The converter also searches for datatype properties with a corresponding SPARQL query and converts them into GraphML statements for menu lists; it acts likewise for object properties, since they also appear in the submenus. For each property listed in a submenu, the number of their instances in the knowledge base is indicated. The XML converter, therefore, also transfers the numerical information to the GraphML notation—including the number of instances of classes. As such, a submenu list contains all attributes describing the class in the ontology.

GraphML is a comprehensive and easy-to-use file format for graphs. Its syntax is based on XML for graph descriptions. A GraphML/XML specification describing a graph is contained in a GraphML file. An easily readable and understandable description of the GraphML facilities can be found at `http://graphml.graphdrawing.org/primer/graphml-primer.html`. The primer describes the language features through examples that are complemented by references to the relevant W3C recommendation documents (e.g., XML, RDFS, etc.).

Consider the GraphML document and the graph shown in Figure 4.31. The document pictured on the left side defines the graph that is displayed on the right side of the figure. In a GraphML document, a graph is, unsurprisingly, denoted by the element `<graphml>`. Nested inside a graph element are the declarations of nodes and edges. A

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml
   xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
   http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

   <graph id="G" edgedefault="undirected">
      <node id="n0"/>
      <node id="n1"/>
      <node id="n2"/>
      <node id="n3"/>
      <node id="n4"/>
      <node id="n5"/>
      <node id="n6"/>
      <node id="n7"/>
      <edge source="n0" target="n2"/>
      <edge source="n1" target="n2"/>
      <edge source="n2" target="n3"/>
      <edge source="n2" target="n4"/>
      <edge source="n3" target="n5"/>
      <edge source="n4" target="n5"/>
      <edge source="n4" target="n6"/>
      <edge source="n6" target="n7"/>
   </graph>
</graphml>
```

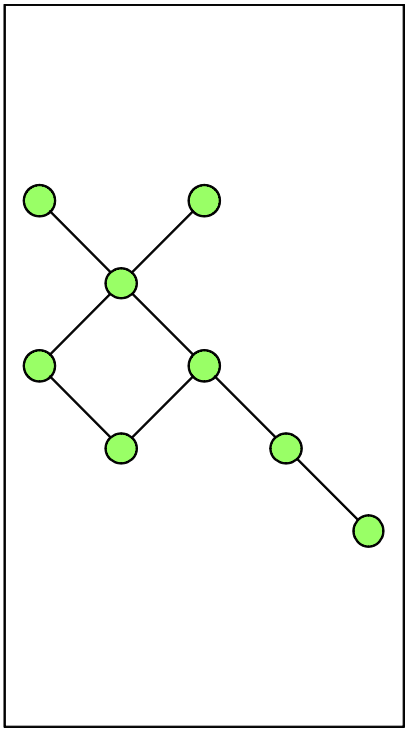


Figure 4.31: The GraphML document specification on the left side describes the graph displayed on the right side of the figure.

node is declared with a `<node>` element, and an edge with an `<edge>` element. One could also define labels for the nodes and edges; GraphML in fact allows many more mechanisms and advanced graph models.

The transformation of RDF/XML triples into GraphML descriptions is straightforward. The transcription of two OWL classes and object properties such as

```
<owl:Class rdf:ID="State">
   <rdfs:subClassOf
      rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:Class rdf:ID="Mountain">
   <rdfs:subClassOf
      rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="isMountainOf">
   <rdfs:domain rdf:resource="#Mountain"/>
   <rdfs:range rdf:resource="#State"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasMountain">
   <owl:inverseOf rdf:resource="#isMountainOf"/>
   <rdfs:domain rdf:resource="#State"/>
   <rdfs:range rdf:resource="#Mountain"/>
</owl:ObjectProperty>
```

generates the following GraphML excerpt, which defines the graph depicted in Figure 4.32 (except for colors and shapes):

```
<graphml
   <graph id="ontologymodel">
      <key id="class" for="node" attr.type="string"/>
      <key id="objectproperty" for="edge" attr.type="string"/>
      <node id="state">
         <data key="class">State</data>
      </node>
      <node id="mountain">
         <data key="class">Mountain</data>
      </node>
      <edge id="hasMountain" source="state" target="mountain">
         <data key="objectproperty">hasMountain</data>
      </edge>
      <edge id="isMountainOf" source="mountain" target="state">
         <data key="objectproperty">isMountainOf</data>
      </edge>
   </graph>
</graphml>
```

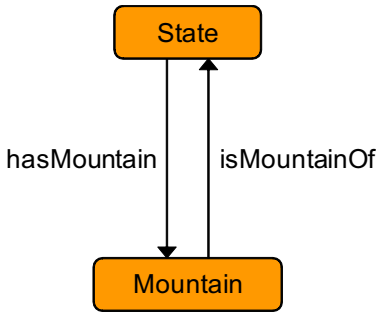


Figure 4.32: The transcription of the two OWL classes "state" and "mountain" and the two object properties "hasMountain" and "isMountainOf" into GraphML results in a simple graph.

The GraphML approach and the Prefuse Visualization toolkit based on GraphML documents are, obviously, well suited for their integration into Semantic Crystal.

5

Retrieval Performance Evaluation

The second key step of this thesis is a thorough evaluation of the interfaces developed in the first step. In fact, to evaluate our two hypotheses regarding the adaptivity barrier and the habitability problem, two evaluations were conducted: *a retrieval performance evaluation* based on test sets and *a comprehensive usability study* with casual end-users assessing our interfaces. The first evaluation provides an answer to our adaptivity hypothesis and, consequently, the second evaluation provides an answer to our habitability hypothesis.

In this chapter, we present the retrieval performance evaluation that we conducted to test the adaptivity hypothesis. Our adaptivity hypothesis predicts that portable NLIs to the Semantic Web with a high quality of retrieval performance can be built with low complexity, since such knowledge bases offer a rich source of semantically annotated information, by moderately restricting the query language of an interface and by extracting the necessary information in order to process natural language queries from the underlying ontology-based knowledge base. The complexity of such query systems can, hence, be reduced to a reasonable developing effort.

To test the hypothesis, we carried out an in-depth test set evaluation in which our NLIs were benchmarked against three existing NLIs with regard to portability and retrieval performance on three different test data sets. The goal was to establish an objective comparative measure of retrieval performance for our interfaces in contrast to three other NLIs. We also aimed at gaining more insight into the advantages and disadvantages of our different NLI approaches.

The following sections describe the experimental setup, the test sets, the methodology, and the results of the evaluation. We used the same three test sets with which the other systems have already been evaluated to enable a comparison. Each test set consisted of both a background knowledge base as well as several hundred English questions with their corresponding logical representation. We ran the questions through three of our interfaces, measured the number of generated formal queries, and then compared the answer sets returned by the formal queries according to the typical information retrieval performance measures.

5.1 Experimental Setup and Methodology

The general evaluation setup and methodology of the retrieval performance evaluation followed the recommendations of Salton and McGill [Salton and McGill, 1983]. To evaluate the quality of the retrieval performance of our interfaces, and to compare them with three other existing approaches, we completed an in-depth test set evaluation. We employed three test sets that had been used as a benchmark in other NLI evaluations, and which therefore served ideally for the comparison of our tools' performance with others'. Specifically, we benchmarked our natural language interfaces NLP-Reduce, Querix, and Ginseng with COCKTAIL by Tang and Mooney (who also generously provided the data sets), with the PRECISE approach by Popescu *et al.*, and with PANTO, a recently developed Semantic Web NLI by Wang *et al.*.

COCKTAIL is an inductive logic programming approach for the task of learning a semantic parser using different learning strategies, i.e., different clause constructors from different learners, in a unifying algorithm [Tang and Mooney, 2001]. The approach was evaluated on the test sets, which were originally implemented as Prolog facts knowledge bases. As described in the related work chapter, PRECISE is an NLI to relational databases allowing users to phrase queries in full English [Popescu et al., 2003]. It uses a max-flow algorithm to generate corresponding SQL queries for the natural language input. The PANTO system is a portable natural language interface to ontologies that, similar to our NLIs, transforms natural language questions into SPARQL queries [Wang et al., 2007]. The transformation is based on parse trees, from which nominal phrases are extracted to form triples, which are, in turn, mapped to the triples in an ontology-based knowledge base.

We ran all natural language queries that came with the three benchmark data sets through NLP-Reduce, Querix, and Ginseng (as described in the next section). Obviously, natural language questions cannot be entered into Semantic Crystal. Therefore, we did not include Semantic Crystal in the performance evaluation. As far as we have observed in the usability study (see Chapter 6), the retrieval performance of Semantic Crystal is of very high quality. To all the queries entered by the subjects participating in the usability study, Semantic Crystal returned the correct answer, which is not surprising when considering that the interface's query language directly maps formal SPARQL statements.

The first goal of the performance evaluation was to discover how many of the English questions that were provided with the three data sets were accepted by the query languages of NLP-Reduce, Querix, and Ginseng. Each accepted and, therefore, processible question generating a corresponding SPARQL query is a *semantically tractable* query [Tang and Mooney, 2001, Popescu et al., 2003]; whether or not the SPARQL translation is correct is irrelevant for the definition of semantic tractability. After assessing the acceptance rates, we then systematically analyzed the questions that were intractable in order

to find common and frequent patterns in natural language questions that were rejected for each of the three interfaces.

The second goal was to measure the retrieval performance of the generated SPARQL queries, thereby deriving the adequacy of the translation and matching for each data set with each of the three systems. Essentially, we executed each SPARQL query with the system that derived the SPARQL statements and compared the returned answer sets to the answers provided by the data set. As the data set also contained corresponding logical queries for each natural language question, we could retrieve the answers by executing the logical queries with a Prolog engine. This enabled us to contrast the returned answer sets of NLP-Reduce, Querix, and Ginseng to the answer set from the original data set. Each answer that did not fully agree with the data set's answer was manually inspected and assessed. Next, the average retrieval performance of each system was measured with standard performance metrics *precision* and *recall*, including the appropriate statistical tests.

We will explain all the measures used in the retrieval performance evaluation after we provide more information about the benchmark data set.

5.1.1 Data Sets

The benchmark data sets were based on the *Mooney Natural Language Learning Data* provided to us by Ray Mooney and his group from the University of Texas at Austin [Tang and Mooney, 2001].[1] We, therefore, would like to thank Ray Mooney and his team for having generously supplied the data on which this evaluation and, in fact, all the examples provided in this thesis are based.

The data comprises three data sets, each supplying a knowledge base, English questions, and corresponding logical queries. They pertain to three different domains: geographical data, job data, and restaurant data. We chose the data sets because of their previous use by other, related projects—a reality that allowed us draw the comparison we wanted to make. In addition, it is in general rather difficult to find good benchmark data sets allowing work with ontology-based NLIs.

To make the original knowledge bases accessible to our ontology-based interfaces, we translated the Prolog knowledge bases to OWL and designed a class structure as the meta model for each of the three domains. The resulting geography OWL knowledge base contained 9 classes, 11 datatype properties, 17 object properties, and 697 instances. The job knowledge base had 8 classes, 12 datatype properties, 8 object properties, and 4141 instances. Finally, the restaurant knowledge base held 4 classes, 5 datatype properties, 8 object properties, and 9749 instances. The ontology models of the three OWL knowledge

[1] `http://www.cs.utexas.edu/users/ml/nldata.html`

bases are represented as graphs in Appendix A (starting on page 171).

Each data set also comprised data-appropriate English questions, which were composed by undergraduate students of the computer science department of the University of Texas in Austin and gathered from "real" people using a Web interface provided by Mooney's research group. There were 877 natural language questions for the geography knowledge base, 620 for the job data, and 251 questions for the restaurant domain. For each question, there was also a corresponding logical representation stated as Prolog terms in the data set. For instance, the logical representation of the question "How many people live in the capital of Georgia?" is:

```
query(A,(population(B,A),capital(B),isLocatedIn(B,C),
              constant(C,state(georgia)))).
```

5.1.2 Data Analysis

When analyzing and interpreting data gathered in an experiment, the issues of validity and reliability should be the underlying principles [Bortz and Döring, 2002] [Nielsen, 1993, Preece et al., 2002]. *Validity* consists of whether an evaluation technique measures what it is supposed to measure, and encompasses both the technique itself and the way it is applied. The *reliability* (or consistency) of an evaluation technique measures whether the same results can be produced if the test is repeated. We have attempted to satisfy both principles in our retrieval performance evaluation.

Our first means for analyzing the resulting data after running all the queries from the three data sets through NLP-Reduce, Querix, and Ginseng is the measure of *semantic tractability* as defined in Popescu *et al.* [Popescu et al., 2003]. It provides an answer to the following question: How many natural language questions did each of our three systems successfully transform into one or more SPARQL queries for each data set? If an interface accepts a question as input and generates a SPARQL query, then the question is said to be *semantically tractable*. At this point, it does not matter if the query produced is appropriate or not. We merely investigate how many questions conformed to the query languages of our NLIs without changing the natural language questions given in the test sets or the interfaces. Semantic tractability is measured by counting the fraction of such questions for each data set.

After executing all SPARQL queries from the semantically tractable questions and storing the results, we analyzed the results with the usual performance measures *precision* and *recall* from the information retrieval literature [Salton and McGill, 1983] [Baeza-Yates and Ribeiro-Neto, 1999]. *Recall* is the extent to which a search engine retrieves all of the items that a user is interested in (i.e., avoiding false negatives), while

precision is the extent to which the tool retrieves only the items of interest (i.e., avoiding false positives). With the precision metric, we can measure the number of correctly produced SPARQL queries that retrieve the correct answer set.

In order to compare our results with COCKTAIL, PRECISE, and PANTO, we slightly adapted the definitions of the two metrics in order to measure the retrieval performance of the semantically tractable questions. Thus, precision and recall are defined as per Tang and Mooney [Tang and Mooney, 2001] and Popescu *et al.* [Popescu et al., 2003] as follows:

> The *recall* of an NLI on the data set is the number of English questions given by the test set that were correctly answered by an NLI divided by the total number of questions:

$$\text{recall} = \frac{\text{number of correct SPARQL queries produced}}{\text{total number of questions}}$$

> The *precision* of an NLI on a data set is the number of English questions given by the test set in which the NLI correctly matches a question to the corresponding SPARQL query divided by the number of questions that the NLI answers (i.e. the semantically tractable questions).

$$\text{precision} = \frac{\text{number of correct SPARQL queries produced}}{\text{number of semantically tractable questions}}$$

Additionally, we used the appropriate statistical tests to determine the significance of the calculated recall and precision values:

- With *ANOVA* (Analysis of Variance), for example, we can determine, if there is statistical evidence to say that the recall and precision values achieved by one of the six different NLI approaches outperforms the other ones.
- When comparing the values of two systems, we used *Student's T-tests* because they check for differences with regard to just two value sets [Rasch et al., 2004].

- To determine whether there is a statistical difference between the NLIs, we based our assessment on the *p-values* (probability values), which are provided as result of ANOVA and T-tests. A p-value less than 0.05 indicates a statistically significant difference, whereas a p-value equal or greater than 0.05 indicates no significant evidence [Sachs, 2004], meaning that, for example, there is no significant difference between the recall and precision values achieved by two or more NLIs.

We will return to these statistical tests when discussing the analysis of our usability study data (cf. Section 6.1.4) and provide more details on the tests there, since they are even more crucial for the argumentation of the usability study's results.

5.2 Results of the Retrieval Performance Evaluation

In this section, we present the results with regard to the semantic tractability and the recall/precision analysis achieved by our NLIs after running the three test sets.

5.2.1 Semantic Tractability

Table 5.1 and the corresponding Figure 5.1 show how many natural language questions are semantically tractable questions for NLP-Reduce, Querix, Ginseng, and PRECISE for the three data sets: geography, restaurant, and job postings. Note that we did not have the numbers for COCKTAIL and PANTO. We see that PRECISE clearly outperforms our NLIs in the job and restaurant data, but does not reach statistic significance (ANOVA, p = 0.434). Querix achieved the best result for the geography data, slightly outperforming PRECISE. Ginseng achieved the lowest rates for the geography and the job data; NLP-Reduce is always somewhat better than Ginseng.

	Geography	(877)	Restaurant	(251)	Job	(620)
NLP-Reduce	515	58.72%	207	82.47%	228	36.77%
Querix	696	79.36%	158	62.95%	243	39.19%
Ginseng	351	40.02%	196	78.09%	179	28.87%
PRECISE	680	77.5%	244	97%	546	88%

Table 5.1: Number of semantically tractable queries achieved by NLP-Reduce, Querix, Ginseng, and PRECISE for the geography, restaurant, and job data. The total numbers of natural language questions are indicated in brackets.

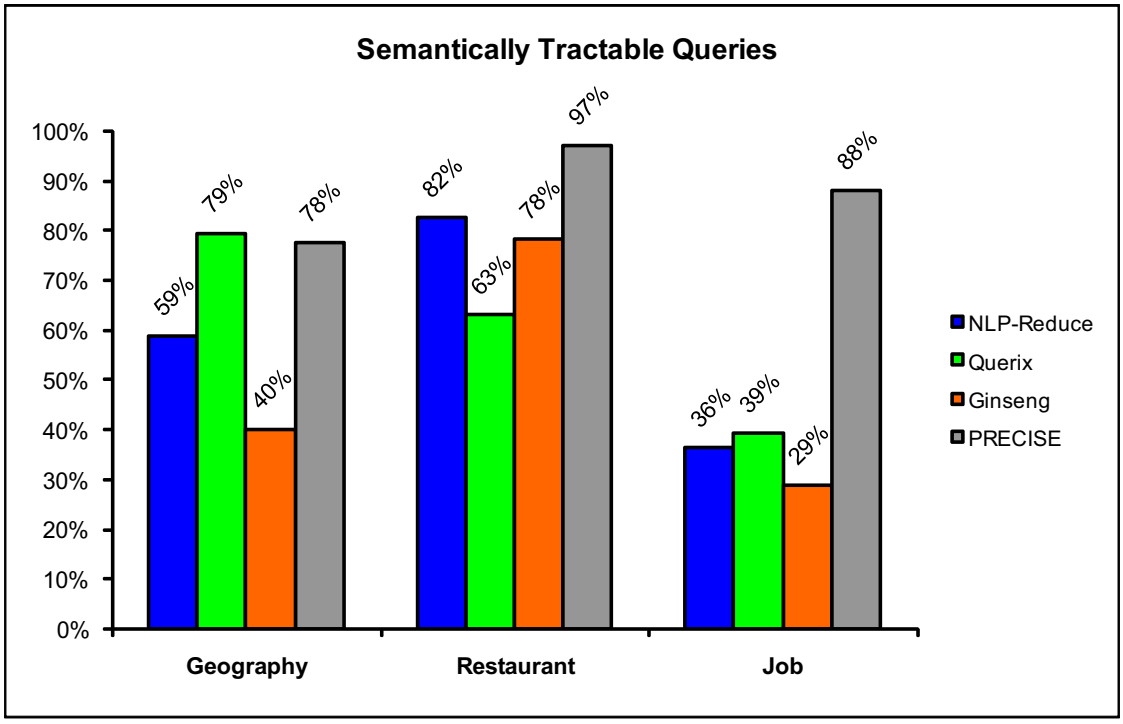


Figure 5.1: Semantically tractable queries achieved by NLP-Reduce, Querix, Ginseng, and PRECISE for the geography, restaurant, and job data (given in percent).

The results for the semantic tractability are not surprising. We ran all questions provided by the test data without changing them, but, since Querix and Ginseng have restricted query languages, some of the full natural language test set questions could not be entered into the two systems. The low semantic tractability rates of Ginseng reflects its limited query language. Nevertheless, we were surprised by the high number of semantically tractable restaurant questions that were accepted by Ginseng. When looking at these questions, we found that they were rather plain and, particularly, stereotypical formulations, such as "Give me a restaurant in San Francisco that serves French food." or "How many Italian restaurants are there in the Bay Area?". As such, Ginseng's static grammar was able to handle many sentence structures of the restaurant questions. PRECISE, however, does not restrict the natural language input at all, and therefore achieved high numbers of semantically tractable questions.

Both the geography and the job questions exhibit many comparative and superlative forms of adjectives, which cannot be processed by NLP-Reduce, and therefore decrease its semantic tractability rate. Furthermore, negations and quantifiers (e.g., "most," "at least") minimize the number of semantically tractable questions for both NLP-Reduce and Querix. The low number of semantically tractable job questions achieved by Querix

is due to several hundred questions beginning with "Are ...," "Show ...," 'Tell...," "List ...," or "Where ...," which are not contained in Querix's set of possible sentence beginnings. As mentioned above, the questions querying the restaurant data are rather simple and stereotype. They, therefore, produce relatively high semantic tractability results for all interfaces. However, a considerably high number of questions starting with "Where ..." once more produce the low results of Querix.

We think that semantic tractability results of our three NLIs are good, especially considering the limited query languages of Querix and of Ginseng in particular. As such, the number of intractable questions is not too serious, since end-users can formulate their questions according to the query language allowed by each search interface without feeling too restricted.

5.2.2 Recall

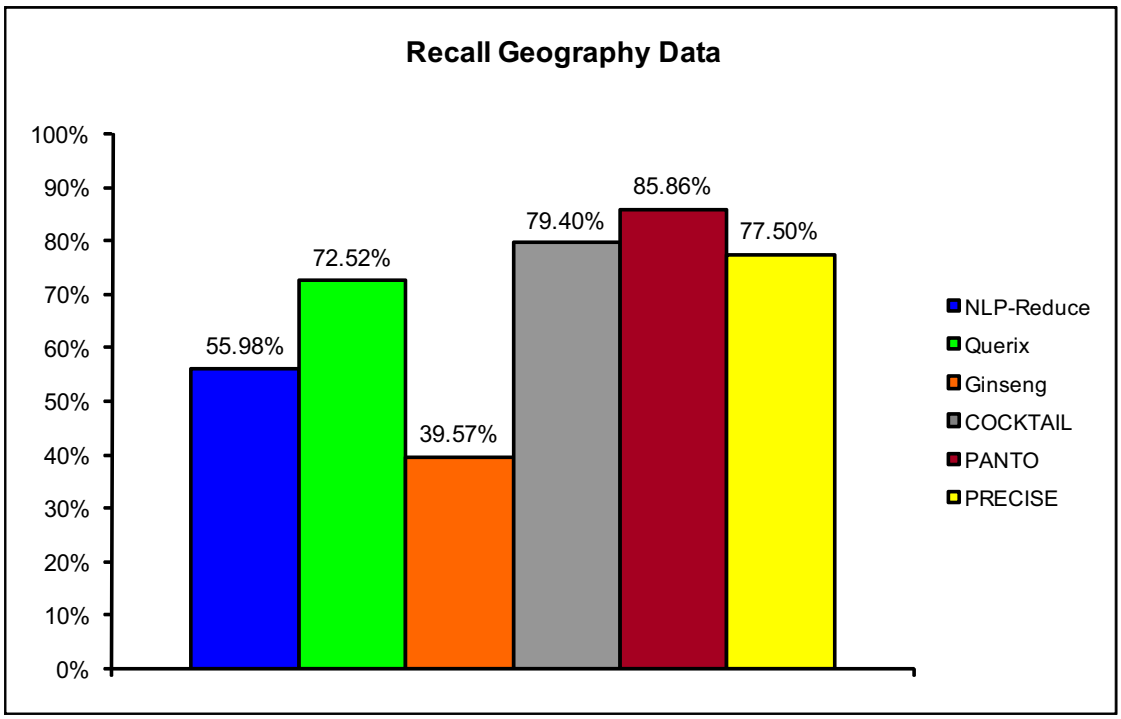


Figure 5.2: Recall achieved by the six systems for the 877 queries of the geography data.

The next step was to execute the SPARQL queries of the semantically tractable questions of each data set through our three NLIs, and to measure their retrieval performance in terms of recall and precision in order compare them with the values achieved by

COCKTAIL, PRECISE, and PANTO. The results for the recall values of all interfaces with the three data sets are shown in Figures 5.2, 5.3, and 5.4.

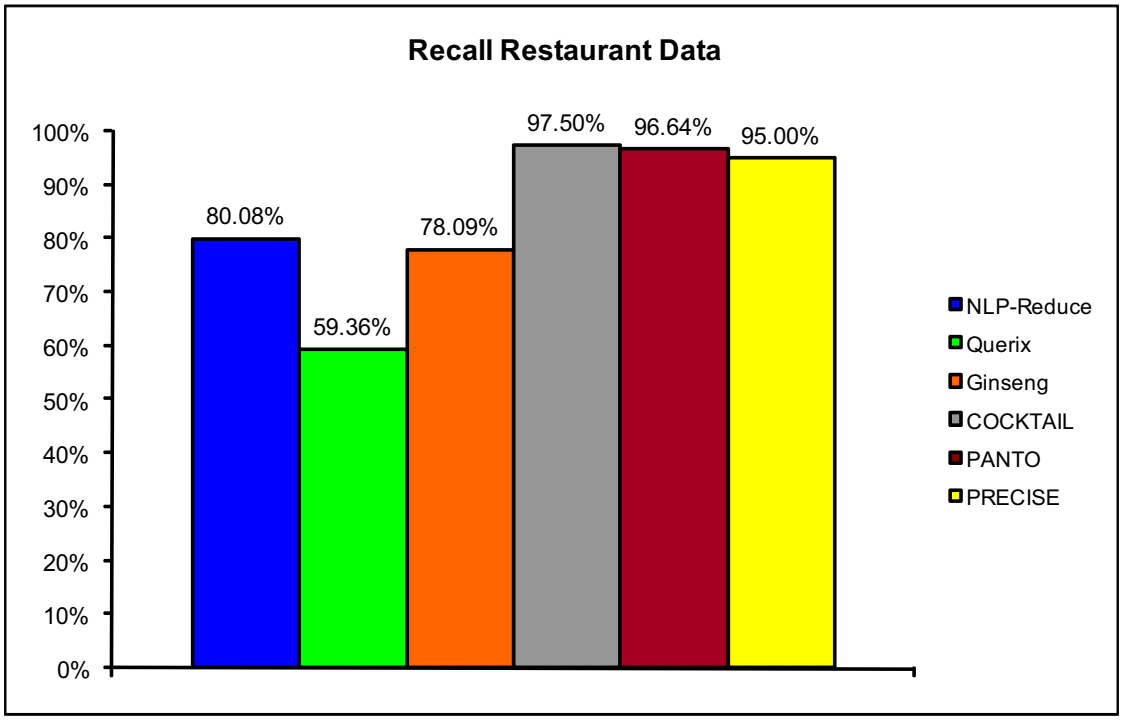


Figure 5.3: Recall achieved by the six systems for the 251 queries of the restaurant data.

We see that PANTO performs the best recall for the geography and the job data, and COCKTAIL for the restaurant data. PRECISE ranks second once and third twice. Our three NLIs are noticeably outdistanced in each data set with regard to recall. The results, however, are as expected, for recall is directly dependent on the number of intractable questions; call to mind that the total number of questions appears in the fraction of the recall definition. As such, the recall values essentially mirror the semantic tractability results, which were lower for our restricted systems—particulary for the job questions. We can thus conclude that both COCKTAIL and PANTO must have a relatively high semantic tractability and, therefore, highly non-restrictive query languages. None of the results for recall reach a statistical significance level; the p-value for the average recall of each system with each data set calculated by ANOVA is 0.380.

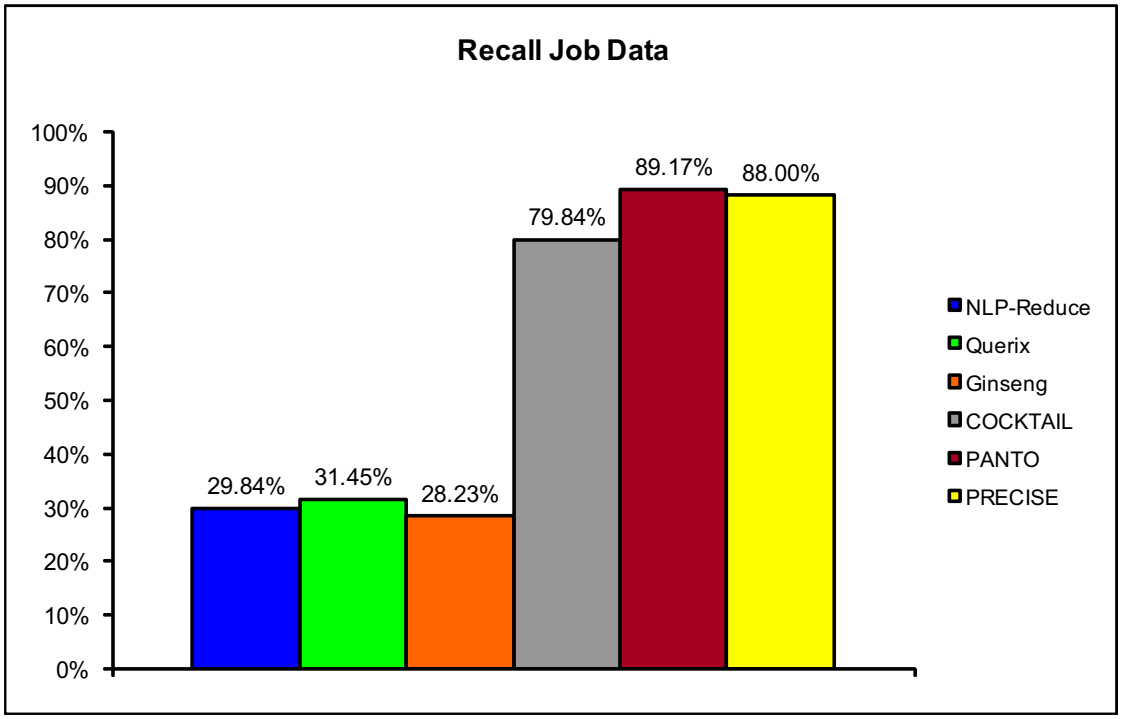


Figure 5.4: Recall achieved by the six systems for the 620 queries of the job data.

5.2.3 Precision

The results for the precision values that were achieved by each of the query systems with each of the data sets are presented in Figures 5.5, 5.6, and 5.7. Most evident is the fact that PRECISE makes no mistakes, meaning that each SQL query is correct and retrieves the correct answer to the natural language question. PRECISE outperforms all other systems by achieving 100% precision with each data set and thus nearly achieving statistical significance (ANOVA, $p = 0.056$).

The second striking outcome is that Ginseng consistently achieves second place, and also achieves 100% precision for the restaurant data. This performance is due to the restricted vocabulary and the controlled sentence structures, which can be mapped incrementally to appropriate SPARQL statements. Hence, almost whenever Ginseng can parse a question, it will translate it to a correct formal query. With regard to precision, the restrictive query language of Ginseng is beneficial.

NLP-Reduce performs with similar efficiency, reaching the upper nineties for the geography and restaurant data, while "only" reaching 81.14% precision for the job data. When browsing the false SPARQL mappings, we found that most mistakes were due to NLP-Reduce's inherent design. The simple approach does not parse the input questions

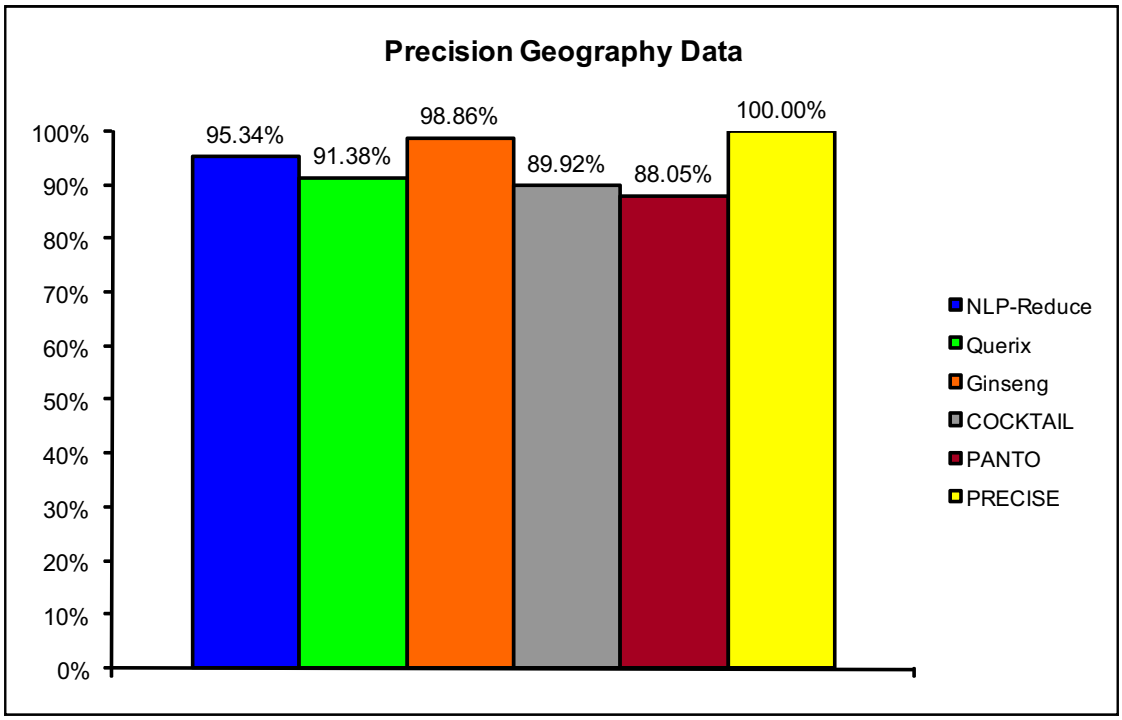


Figure 5.5: Precision achieved by the six systems for the 877 queries of the geography data.

and, as such, completely ignores sentence structures and semantic dependencies that exist between the constituents of a sentence. Some of the job questions possess complex structures and a rich vocabulary, such as "What jobs are there doing computer graphics on silicon graphics machines?", "Give me the jobs for people in Austin that want to program in Lisp.", or "Which jobs in Houston offer for students fresh out of college in networking?". These sentences cannot be transformed to correct SPARQL queries by NLP-Reduce, therefore resulting in lower precision for the job data. As a matter of fact, the last example is a grammatically incorrect sentence.

The job questions' complexity is also responsible for the relatively low precision achieved by Querix. The small set of heuristic patterns for identifying and joining the triples in the questions are insufficient for handling the complex sentence structures. Another problem for Querix occurred in the geography data because of questions containing noun phrases such as "the state California" or "the Mississippi river," where the heuristic triple patterns consistently missed the second noun, therefore resulting in inadequate SPARQL queries. Some mistakes were caused by the Stanford parser, which shows an average accuracy performance of 83.43% [Hempelmann et al., 2005]; its speed, however, is first-rate.

While NLP-Reduce and Querix encounter difficulties with the job data, COCKTAIL

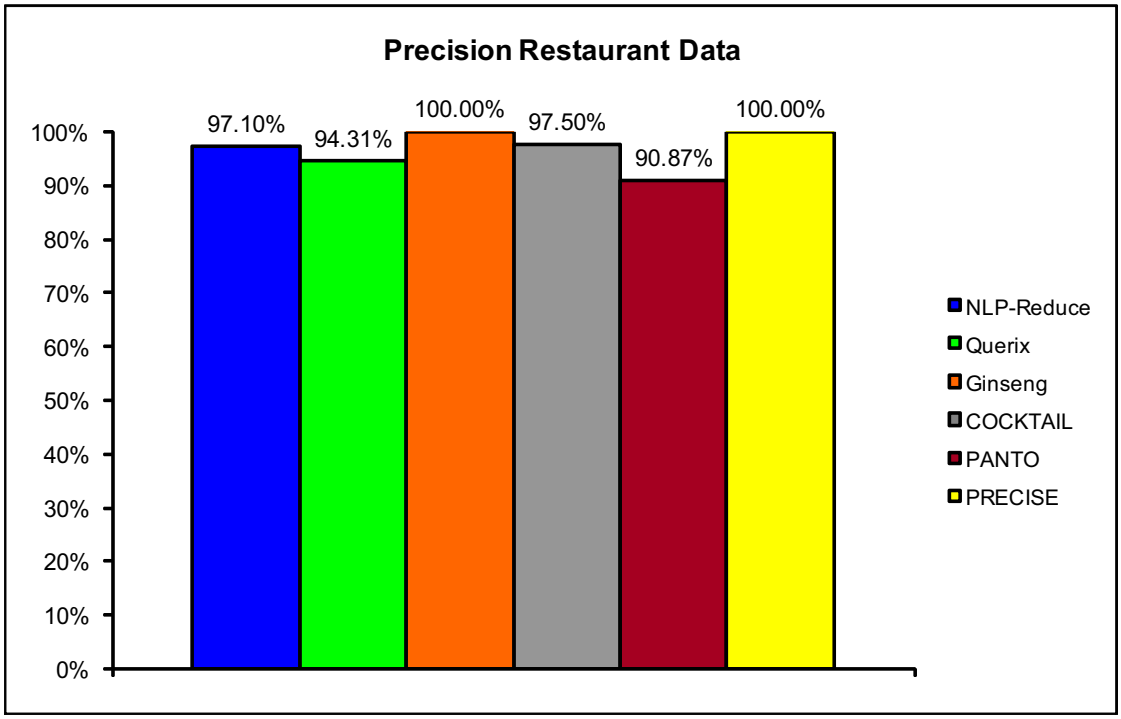


Figure 5.6: Precision achieved by the six systems for the 251 queries of the restaurant data.

achieves a precision of 93.25% and performs even better for the restaurant data (97.50%). However, it performs worst among the six competitors with the geography data.

	Average Recall	Average Precision
NLP-Reduce	55.30%	91.19%
Querix	54.44%	88.65%
Ginseng	48.63%	98.87%
COCKTAIL	85.58%	93.56%
PANTO	90.56%	88.35%
PRECISE	86.83%	100%
p-value (single factor ANOVA with 6 levels)	0.038	0.056

Table 5.2: Average recall and precision achieved by the six systems for the three data sets from the domains of geography, restaurants, and jobs.

The **average precision** performed by PANTO for all data sets is the poorest of all interfaces if we dare to refer to an average precision of 88.35% as poor (see Table 5.2).

PANTO, on the other hand, shows the best **average recall** for the three data sets (90.56%), thereby illustrating the unavoidable tradeoff between recall and precision. This is due to the tendency for precision to decline as recall increases in information search tasks [Buckland and Gey, 1999].

The overall precision performance of all six interfaces with a total of 1748 questions from three different domains is excellent, ranging from 80.25% for Querix on the job data to 100% precision achieved three times by PRECISE and once by Ginseng.

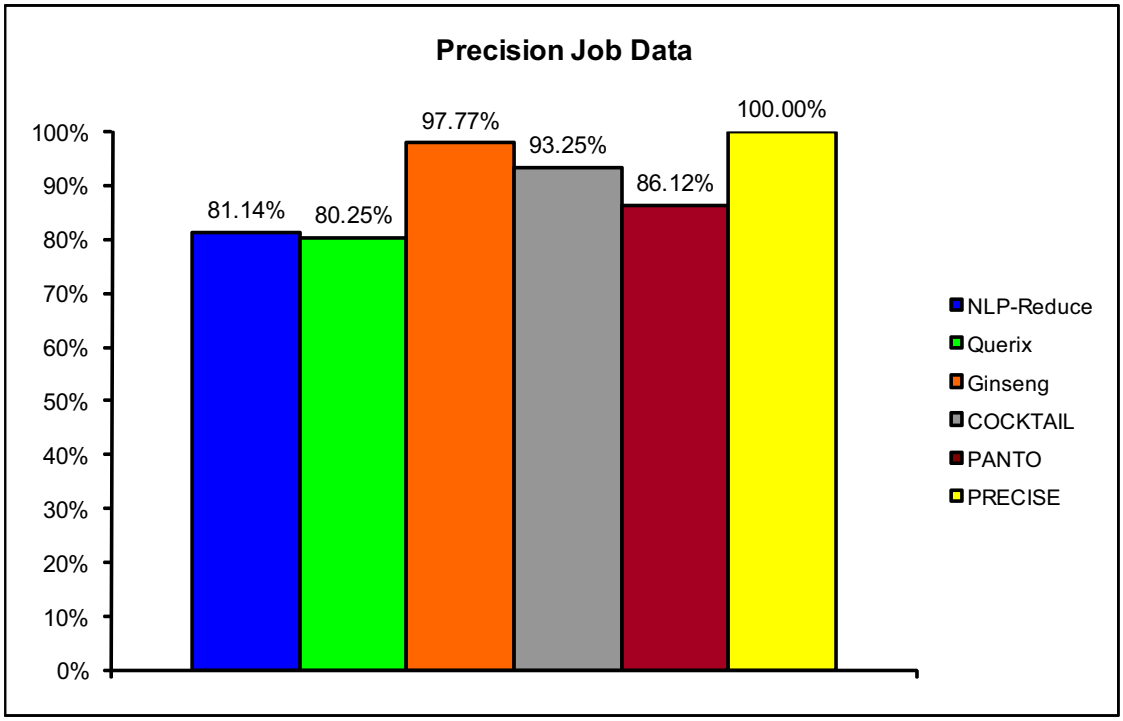


Figure 5.7: Precision achieved by the six systems for the 620 queries of the job data.

5.3 Discussion of the Most Remarkable Results

In spite of simple designs applying basic pattern-matching algorithms and low-level NLP techniques, our three natural language search interfaces NLP-Reduce, Querix, and Ginseng performed very well in the retrieval performance evaluation. Their average recall values are evidently lower than the precision values (cf. Table 5.2), which is a logical consequence of the fact that the recall metric incorporates the number of semantically tractable questions. The semantic tractability accomplishments of controlled natural language interfaces are inherently lower due to their restriction of the query input. In our

evaluation, however, we used full natural language questions as constructed by students and Web-interface users, and retained them unchanged. The precision performance of our three NLIs can, therefore, be considered as very high but still improvable. As such, we can call NLP-Reduce, Querix, and Ginseng both portable or domain-independent and well-performing in information seeking tasks.

The declining recall values from NLP-Reduce to Querix and Ginseng inversely reflect the increasing degree of formality and restrictiveness of the systems' query languages. However, we expected the differences to be greater. Linguistic phenomena such as negation, quantifiers, and comparative/superlative adjectives depressed the recall of NLP-Reduce. The fixed set of sentence beginnings did the same for Querix, whereas the stereotypical questions of the restaurant data increased Ginseng's recall. On the other hand, the mapping of the precision performance values to the increasing degree of restrictiveness in the Formality Continuum fails, since the average precision achieved by the least restrictive interface, NLP-Reduce, outperformed the precision of Querix. The complexity of the sentences in the job data as well as the compound nominal phrases in the geography data decreased the precision of Querix and revealed some weaknesses of the simple triple pattern matching procedure. With every new data set, we would probably find other data-inherent or domain-typical phenomena influencing the retrieval performance of our NLIs—a problem also diagnosed by Tang and Mooney [Tang and Mooney, 2001].

The adaptivity hypothesis has largely been corroborated by our three NLIs and, at least partially, by the benchmark systems COCKTAIL, PRECISE, and PANTO:

> Yes, we can develop portable NLIs to the Semantic Web with low complexity and with a high quality of retrieval performance by extracting the necessary information to analyze and map natural language questions to formal SPARQL queries from the underlying ontology-based knowledge bases and by slightly restricting a natural query language.

The ontology-based knowledge bases seem to offer a considerable amount of semantic information, particularly if they are enhanced by common NLP methods such as synonym expansion. As such, NLIs with slightly controlled query languages offer a convenient as well as reliable means of querying access to the Semantic Web and, hence, a realistic potential for bridging the gap between the formal logic of the Semantic Web and casual end-users.

6

Usability Study

The second evaluation we conducted in order to test the habitability hypothesis comprises a usability study. It is intended to provide an answer to our habitability hypothesis, which proposes that some structure should be imposed on casual end-users when formulating queries with a search interface in order to guide but not overly control (and hence alienate) them. Therefore, our assumption is that the best query language solutions will lie somewhere towards the middle of the Formality Continuum.

Specifically, the goal of the usability study was to investigate how useful our three natural language query interfaces NLP-Reduce, Querix, and Ginseng were in order to find data in Semantic Web knowledge bases in comparison with themselves and in comparison with the formal query approach Semantic Crystal. We, additionally, aimed at gathering the data necessary for inferring which degree of naturalness or formality and guidance in a query language finds most approval with casual end-users. We absolutely wanted to discover what query language preferences real-world users possess. As such, the study can contribute to the general discussion on whether NLIs are useful from the end-users' perspective.

After running several preliminary usability experiments and gaining crucial experience with user experiments, we conducted a comprehensive and thorough usability study, in which we benchmarked our four systems against each other with 48 users. This chapter presents the results and accomplishments of this last usability study; the discussions of the preliminary evaluations can be found in Bernstein *et al.* 2005a [Bernstein et al., 2005a], Bernstein *et al.* 2005b [Bernstein et al., 2005b], Bernstein and Kaufmann 2006 [Bernstein and Kaufmann, 2006], and Kaufmann and Bernstein 2007 [Kaufmann and Bernstein, 2007].

In the usability study presented here, casual end-users should test and assess the usability of each of the four systems and, in particular, their query languages. As such, we let casual end-users perform the same retrieval tasks with each of the four tools in order to discover which query language they liked best, which they liked least, and why.

Furthermore, we examined the time they spent to perform the tasks, how many queries they required to find the requested information, and how successful they were in finding the appropriate answers with each system.

To provide the range of query languages and their features required by our four interfaces, we summarize them here:

- NLP-Reduce:
  keywords, sentence fragments, and full sentences
- Querix:
  full sentences that must begin with "Which," "What," "How many," "How much," "Give me," or "Does" and end with a question mark or full stop
- Ginseng:
  predetermined, fixed, controlled, and menu-based words/sentences akin to English
- Semantic Crystal:
  graphically displayed, clickable, formal query language

6.1 Experimental Setup and Methodology

The overall design of our benchmark evaluation followed the methods proposed by Nielsen [Nielsen, 1993] and Rauterberg [Rauterberg, 1991]. As the goal of our test situation was to evaluate different interface and query language alternatives, we performed a deductive benchmark test. Our evaluation employed between-subjects testing as well as within-subjects testing in order to avoid biases and, therefore, the distortion of the results. Before running the actual experiment, we conducted three pilot tests; two are suggested by the literature. This way, flaws in the test design could be identified and eliminated.

6.1.1 Subjects

To benchmark the four interfaces in a controlled experiment with real-world casual users, we promoted the usability study on the Web sites of our department and the university. Additionally, we promoted the study by billboard advertisements, which we distributed randomly across the city of Zurich. We ended up with 48 subjects almost evenly distributed over a wide range of backgrounds and professions: bankers, biologists, computer scientists, economists, game programmers, housewives, journalists, language

teachers, mechanical engineers, musicians, pedagogues, psychologists, secretaries, sociologists, veterinarians, video artists, and unemployed persons to name most of them. The participants were composed of 19 males and 31 females. There was a normal distribution of age ranging from 19 to 52 years with a mean of 27.6 years. As such, our subjects represented the general population of casual search interface end-users. Having 48 users also enabled us to cover each possible order of the four systems not just once but twice, a fact that increases the overall statistical significance of the results (see Section 6.2 below).

The subjects involved in our benchmark evaluation were given an reward, i.e., a monetary experimental fee, for their work to ensure a correct incentive-set, which should not be underestimated [Croson, 2005]. When testing with humans, it is, furthermore, important to take ethical aspects into account [Nielsen, 1993]. We had to make certain that the test users were aware that the query interfaces were being tested and not the users, an important issue that can severely influence test results [Rauterberg, 1991]. We also had to ensure the subjects' anonymity and the existence a confidential data storing.

6.1.2 Tasks / Experimental Procedure

For each interface, the users were asked to perform the same tasks: They had to reformulate four questions presented to them as sentence fragments into the respective query language required by the four systems and then enter the questions into the interfaces. The four questions were predominantly the same for each system, but slightly altered in order to increase interest for the users. For example, one question was "area of Alaska?" given for NLP-Reduce and "area of Georgia?" for Querix etc. The four question templates were:

- area of Alaska?
- number of lakes in Florida?
- states that have city named Springfield?
- rivers run through state that has largest city in US?

In principle, each interface is able to answer all four queries. Each system does, however, "stumble" over one of the queries such that, for example, more than one query is needed to retrieve the correct result, or one of the words in the question templates cannot be recognized by the interface and must either be replaced with another word or omitted altogether. We chose the query templates very carefully to provide a maximally fair competition for the four system. For every user, we changed the order in which the interfaces were presented as well as the order of the queries for each system so as to prevent learning effects from influencing the results.

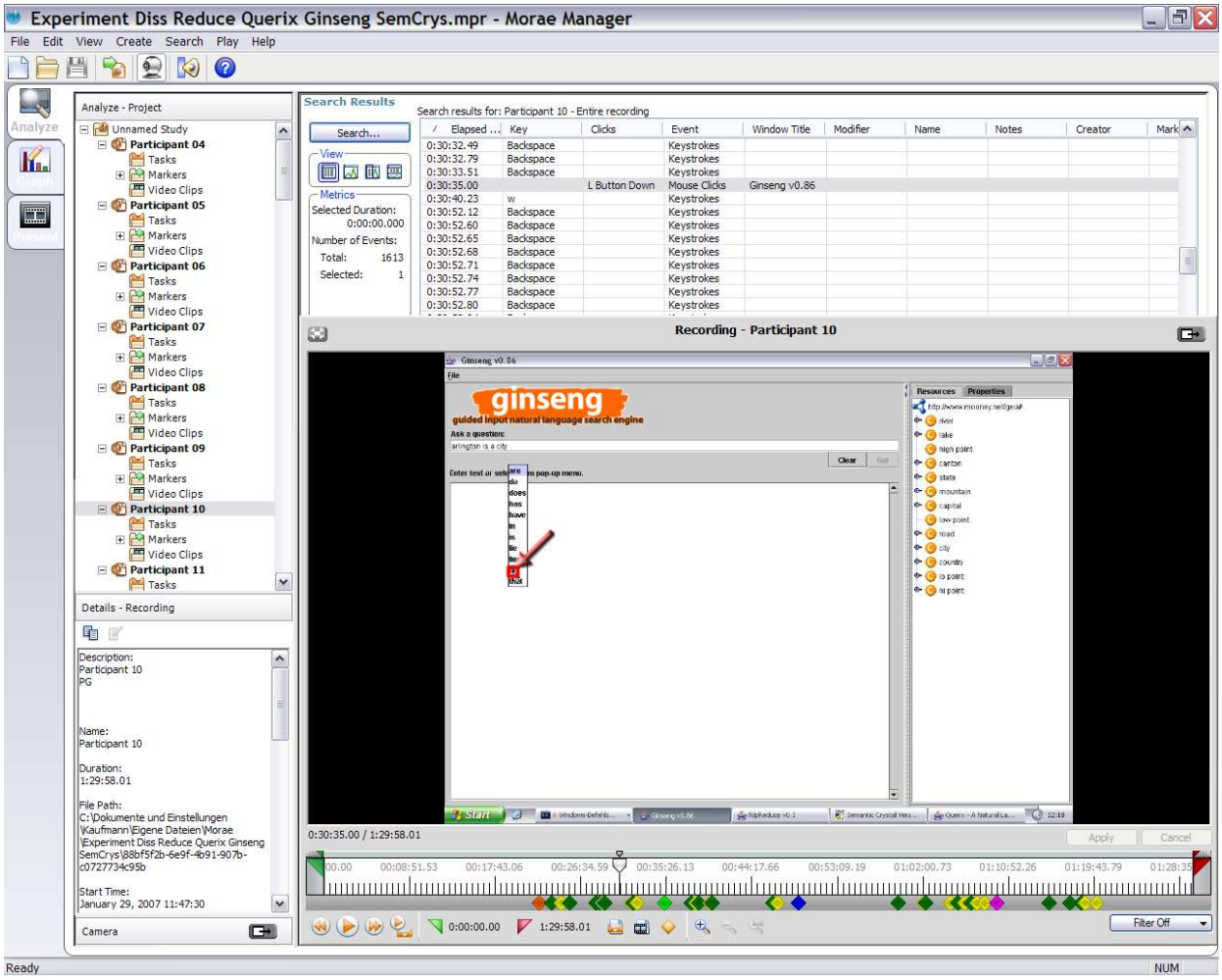


Figure 6.1: The Morae Software for usability testing allows an experimenter to remotely watch and annotate the desktop of a test user performing experimental tasks, and then analyze the results with the software's manager tool.

After completing the questions with each interface, the users were asked to answer the *System Usability Scale* (SUS) questionnaire (provided in appendix B on page 175). SUS is a standardized usability test by Brooke [Brooke, 1996] containing ten standardized questions (e.g., "I think that the interface was easy to use," "I think that I would need the support of a technical person to be able to use this system," etc.). Each question is answered on a 5-point Likert scale establishing a person's impression regarding a user interface. The test covers a variety of usability aspects such as the need for support, training, as well as complexity, and has proven to be very useful when investigating the usability of interfaces. The result of the questionnaire is a value between 0 and 100,

where 0 signifies that a user found a system absolutely useless and 100 that a user found a system optimal. As usability is not an absolute criterion, the resulting SUS score can only be understood when comparing it with others, which was the case in our study.

After testing and judging all interfaces, users were explicitly asked to fill in a comparison questionnaire in which they were asked which NLI they liked best and which one they liked least; they were also asked analogous questions regarding the query languages (see appendix C on page 177). We also asked them about the motivations for their choices. At the end of the experiment, people were requested to answer a number of demographic questions, such as age, gender, profession, knowledge of informatics, knowledge of linguistics, knowledge of formal query languages, and knowledge of English (listed in appendix D on page 179).

At the beginning of each experimental run, the test user was given, on paper, all information and instructions concerning the experiment. This written form assured that every user was provided identical information and instructions. At first, the purpose of the test was explained to the test users. Then, the tasks were stated; the pilot tests granted clarity to the task descriptions. We also ensured that each test user knew that he/she could interrupt or abort the experiment anytime. The complete instructions, including all questionnaires as used in the experiment, are attached in appendix E (starting on page 181).

To provide an introduction to the query languages of the interfaces, users were given 1-page instructions for the three NLIs and 2-page instructions for Semantic Crystal. Hence, the procedure of the experiment for each user was the following:

1. read some introductory notes on the overall experiment,

2a. read instructions on the query language of the first interface,

2b. reformulate, enter, and execute four queries with the first interface,

2c. fill in the SUS questionnaire for the first interface,

3. proceed by repeating steps 2a to 2c with the second, third, and fourth interface,

4. fill in the comparison questionnaire,

5. and finally provide answers to the demographic questions.

The overall experiment took about 45 to 60 minutes for each subject. Using the Morae Software,[1] we were able to remotely record any desktop activity of the users as well as log and time each of their key entries and mouse clicks. An observer can also annotate important incidents while an experiment is "on air." All activities and annotations can be analyzed and visualized with the software's manager tool. Figure 6.1 shows a printscreen of the Morae Manager with the recorded desktop of a subject.

6.1.3 Data Set

The usability study was based on a knowledge base containing geographical information about the US from the *Mooney Natural Language Learning Data* by Tang and Mooney [Tang and Mooney, 2001].[2] We chose the data set for the usability study because it covers a domain that can easily be understood by casual users and does not demand expert knowledge [Bernstein et al., 2004]. To make the knowledge base accessible to our four interfaces, we translated it to OWL and designed a simple class structure as a meta model (as described in Section 5.1.1).

6.1.4 Data Analysis

The data we collected in the usability study was analyzed quantitatively as well as qualitatively. For the quantitative analysis, we used the SUS scores and the usual statistical methods ANOVA, T-test, and Mixed Linear Regression Models as available in the R-Software[3] (a free software environment for statistical computing and graphics) and its lme4-package[4] (linear mixed-effects models using S4 classes):

- *ANOVA* or Analysis of Variance is a statistical method of checking whether a relationship exists between two or more data sets. It is like a T-test (see below) conducted simultaneously across multiple data sets, and essentially indicates whether the results from an experiment were due to random chance or not. With ANOVA we can, for example, determine if there is a significant difference between the time our subjects needed to fulfill the tasks they were given in the experiment with one system and the time they required with the other three systems. When comparing the measured times needed by each subject with the four interfaces using ANOVA, we can identify one *independent variable* or *factor* "interface" and, hence, we have a *single factor ANOVA* (also called *one-way* ANOVA) with four levels (i.e., the four

[1] http://www.techsmith.com/morae.asp
[2] http://www.cs.utexas.edu/users/ml/nldata.html
[3] http://www.r-project.org/
[4] http://stat.ethz.ch/CRAN/

values of the factor "interface," which are the four interfaces NLP-Reduce, Querix, Ginseng, and Semantic Crystal).

- In cases where we compared the results of only two systems, we used *Student's T-tests*. Given two data sets, each characterized by its mean, standard deviation, and number of data points, we can use a T-test to determine whether the means are in fact distinct or not.

- For statistical tests such as the T-test or ANOVA, the R-Software outputs *p-values* (probability values). A p-value measures the significance of a statistical test and indicates whether there is statistical evidence to suggest that one measurement set differs significantly from another measurement set. The p-value is a value between 0 and 1. A p-value of 0.05 means that, if one claims that a difference exists between two data sets, there is an error rate of 5% that the difference relies on random chance alone. The smaller the p-value is, the safer it is to say that there is a difference between two or more data sets [Rasch et al., 2004]. Usually, a p-value smaller than 0.05 indicates statistical significance [Sachs, 2004], meaning that two or more data sets do not differ by chance, but by statistical evidence. Consequently, a p-value equal or greater than 0.05 indicates no level of significance. Two or more data sets that are associated with a p-value of 0.71, for example, will not be considered to be of statistical difference.

- Additionally, we used *Mixed Linear Regression Models* to analyze the data collected from the usability study. Mixed linear models are statistical regression models to model means, variances, and covariances in data. Basically, linear regression is a method that models the relationship between a dependent variable (also called *response variable*) and independent variables (also called *explanatory variables* or *regressors*) such that the independent variables have some influence or impact on the outcome of the dependent variable. With Mixed Linear Regression Models, we can, for example, find out if the independent variables *knowledge of informatics*, *knowledge of linguistics*, *knowledge of formal query languages*, and *knowledge of English* significantly influenced the dependent variable *time*, which was needed to reformulate and enter the queries into the four systems. Statistical evidence is again indicated by a p-value less than 0.05, and provided in the lme4-package of the R-Software [Faraway, 2005, Faraway, 2006].

When qualitatively analyzing the data we collected with the comparison questionnaires, we looked for patterns for categorization and peculiar incidents [Preece et al., 2002]. Additionally, we tried to satisfy the internal as well as the external validity [Bortz and Döring, 2002] when interpreting the results and drawing conclusions.

6.2 Results of the Usability Study

6.2.1 Time and Query Reformulation

The results concerning the **time that the users spent** to reformulate the queries in our usability study are summarized in Table 6.1.

	average time for all 4 queries	average time per query
NLP-Reduce	2 min 39 sec	23.54 sec
Querix	4 min 11 sec	29.31 sec
Ginseng	6 min 06 sec	34.82 sec
Semantic Crystal	9 min 43 sec	89.53 sec
p-value (single factor ANOVA with 4 levels)	1.56e-26	4.91e-40

Table 6.1: Results of the average time that users needed to reformulate all four queries with each interface and the average time they spent per query with each interface. The p-value was calculated by a single factor ANOVA.

Most strikingly, our results are more than highly significant (a statistical significance of $p < 0.05$), which is due to the high number of users and the double coverage of every possible interface, as well as query order. The first column shows that users were by far the fastest when entering the four queries with NLP-Reduce (p = 1.56e-26). This outcome is obvious as the query language of NLP-Reduce, which can be full sentences, sentence fragment, or just keywords with no restrictions, imposes least constraints on the user and allows the queries to be entered with fewest words. Users spent most time when working with Semantic Crystal, demonstrating that the intellectual burden of composing semantically and syntactically appropriate formal queries lies exclusively with the user, whereas the other three systems carry some of the burden themselves. The linearly increasing average time that was spent per query well mirrors the increasing degree of formality and control of the interfaces' query languages (see the Formality Continuum in Figure 6.2 on page 143).

In Table 6.2 the **average number of queries** to find answers to the four question fragments and the success respectively the failure of these queries are presented. We can see in column 1 that it took users 7.02 queries on average to find an answer to the four questions given in the experiment with Semantic Crystal and 11.06 query trials with Ginseng. NLP-Reduce and Querix lie between these two and close to each other. The high number of query trials in Ginseng is a result of its query language's control, which causes users

	average number of queries	average success rate (biased)	average failure rate (biased)
NLP-Reduce	7.94	69.27 %	30.73 %
Querix	7.75	77.08 %	22.92 %
Ginseng	11.06	63.54 %	36.46 %
Semantic Crystal	7.02	54.86 %	45.14 %
p-value (single factor ANOVA with 4 levels)	3.92e-06	1.06e-05	2.54e-05

Table 6.2: Results of the average number of queries that were required to find the answers for all four queries with each system, and the success/failure rates of the queries from the test users' point of view.

to repeatedly reformulate and execute their queries in a kind of backtracking behavior. The log files revealed that the lowest number of query trials in Semantic Crystal emerged from users giving up and being unwilling to keep trying until an appropriate query was composed.

The **average success and failure rates** in Table 6.2 indicate how many of the four queries retrieved a satisfying answer from the users' perspective (i.e., the user thought that she/he had found the correct answer). Though Semantic Crystal in fact provides more precise answers than its competitors (see also Table 6.3), the success rate of only 54.86% is due to inappropriate and invalid query formulations. The significantly superior success rate achieved by Querix from the users' point of view seems to be due to Querix's answer display. For example, if a user enters a query "How many rivers run through Colorado?", Querix's answer is: "There are 10.", while the other three interfaces show a list with the names of ten rivers and the number of results found. Some users specifically pointed out in the questionnaires that they trusted the natural language answers of Querix more because the linguistic answer created the impression that the system "understood" the query.

When reviewing the recorded log files in order to find out what the success and failure rates were from an objective point of view, we discovered that there is no discrepancy between the subjects' reports and the objective success/failure rates in Ginseng and Semantic Crystal. There was one case, however, in which NLP-Reduce returned an incorrect answer and the user thought it was correct. Astonishingly, Querix produced 34 incorrect answers out of the total of 420 queries posed by all subjects, but the subjects were actually satisfied with the answers. We traced the false impressions of answers such as "There are 25." that stemmed from interface-based confidence. In contrast, the natural language answers apparently created skepticism, since most of the queries entered to double-check previous answers occurred with Querix (i.e., 22), whereas 17 checking

queries were entered with NLP-Reduce, 3 with Ginseng, and none with Semantic Crystal. The number of checking queries almost inversely mirrors the increasing degree of formality from Querix/NLP-Reduce to Ginseng and Semantic Crystal, which leads us to the hypothesis that formality rather than naturalness may also create a notion of trust towards a system.

	average success rate (objective)	average failure rate (objective)
NLP-Reduce	68.75 %	25.00 %
Querix	59.38 %	15.10 %
Ginseng	63.54 %	36.46 %
Semantic Crystal	54.86 %	44.09 %
p-value (single factor ANOVA with 4 levels)	0.072	1.36e-08

Table 6.3: Success and failure rates for queries entered by the users from an objective point of view, meaning that the interfaces actually generated correct or false answers. (The rates do not reach 100% due to typos.)

Additionally, we detected that, in NLP-Reduce, 15 queries did not lead to a satisfying answer due to typos. There were also 15 queries with typos in Querix, and 2 in Semantic Crystal. As such, the **objective success and failure rates** achieved by the test users with all queries and all interfaces are shown in Table 6.3. The results reveal that NLP-Reduce performs best with regard to correct answers from an unbiased point of view; the result, however, is not significant. Querix was even outranked by Ginseng, although Querix appeared to perform best from the users' perspective. The ranking of the objective failure rate results remains the same.

We also examined the **success and failure rates with relation to the time** that was spent entering and reformulating the query fragments. In Table 6.4 we see that, when we relate the success and failure rates to the time it took users to reformulate an enter the query fragments, the vastly superior success rate was obtained by NLP-Reduce and the lowest with Semantic Crystal ($p = 1.47e\text{-}16$). Again, the results confirm that most time is required when working with the interface that shifts the query formulation burden to the user. The failure rates related to the time spent for entering the queries inversely reflect the same result ($p = 1.84e\text{-}08$). Consequently, the success rates of the more formal and, therefore, more precise query languages cannot counterbalance the additional time that is needed to compose queries with them.

Using the **Mixed Linear Regression Model** analysis, we found that *the order in which the interfaces were presented* to a user slightly influenced the time that was spent on query

	average number of successful queries per minute	average number of failed queries per minute
NLP-Reduce	0.87	0.39
Querix	0.73	0.21
Ginseng	0.41	0.23
Semantic Crystal	0.21	0.17
p-value (single factor ANOVA with 4 levels)	1.47e-16	1.84e-08

Table 6.4: Results of the average number of successful and failed queries per minute. A successful query retrieves a satisfying answer, whereas a failed query does not retrieve a satisfying answer—from the subject's point of view.

reformulation: If a tool was presented last, users spent an average of 37.5 seconds more per query than if the tool was presented first ($p = 0.019$). This finding contradicts the general belief that users tend to become increasingly impatient as an experiment proceeds [Nielsen, 1993, Preece et al., 2002]. On the recorded desktop videos, it looked as if the users were eager to "get it right" during the last iteration of the experiment.

The *order of the four queries* and *the knowledge of informatics*, *linguistics*, *formal query languages*, and *English* did not significantly affect the time. While there was also no correlation between the variable *gender* and the average time spent per query, the variable *age* did: With every year a user's age grows, the average time to reformulate a query increased by 3.30 seconds ($p = 0.010$).

6.2.2 System Usability Scores

Table 6.5 contains the **results of the SUS questionnaires**. Recall that the SUS score is a value between 0 and 100, where 0 signifies that a user found a system absolutely useless and 100 that a user found a system optimally useful. Querix achieved the highest average SUS score, 75.73, and significantly outperformed the other three interfaces ($p = 7.36e\text{-}17$). The graphical query interface Semantic Crystal was not highly appreciated, which is reflected in the average SUS score of 36.09. NLP-Reduce and Ginseng achieved similar SUS scores somewhere in the middle of the other two NLIs; their scores do not significantly differ from each other (paired, one-tailed T-test: $p = 0.356$).

	average SUS score
NLP-Reduce	56.72
Querix	75.73
Ginseng	55.10
Semantic Crystal	36.09
p-value (single factor ANOVA with 4 levels)	7.36e-17

Table 6.5: Results of the SUS questionnaires. The System Usability Score is a value between 0 and 100, where 0 signifies that a user found a system absolutely useless and 100 that a user found a system optimal.

6.2.3 Interface and Query Language Comparison

Upon seeing the SUS results, it is no surprise that 66.67% of the users liked the Querix interface best and only 2% liked it least, even if this result is not significant (columns 1 and 2 in Table 6.6). Querix obtained almost the same feedback for its query language in particular, this time reaching statistical significance with p = 0.0075 (columns 3 and 4).

	interface liked best	interface liked least	query language liked best	query language liked least
NLP-Reduce	12.50 %	25.00 %	18.75 %	25.00 %
Querix	66.67 %	2.08 %	60.42 %	4.17 %
Ginseng	6.25 %	12.50 %	16.67 %	12.50 %
Semantic Crystal	14.58 %	60.42 %	4.17 %	58.33 %
p-value (single factor ANOVA with 4 levels)	0.297	0.297	0.0075	0.0075

Table 6.6: Results of the comparison questionnaires, in which the test users indicated which interface and query language they liked best as well as which interface and query language they liked least.

Even though 60.42% of the users disliked Semantic Crystal as a query interface when comparing it to the other three NLIs, a surprising portion of 14.58% assessed Semantic Crystal as their favorite interface. The graphically displayed knowledge base was explicitly found useful by five users. Only 12.50% liked NLP-Reduce best, and 6.25% Ginseng. With respect to the query language, the results are different: here, the query language of Semantic Crystal received the lowest rating (4.17%), and the query languages of NLP-Reduce (18.75%) and Ginseng (16.67%) were clearly preferred, showing the same ranking as the results of the SUS scores.

When viewing the results of *query language liked least* (column 4 in Table 6.6), the keywords provided by NLP-Reduce were disliked twice as much (25.00%) than the controlled query language of Ginseng (12.50%). We can, therefore, hypothesize that the full freedom of keyword-based query languages is less suitable for casual end-users, since it does not support the user in the process of query formulation. The overall preference for Querix may further reflect this query language tradeoff between freedom that can produce confusion and control that can enable guidance.

The **regression analysis** showed that with each second spent more with a system, the SUS score dropped by 0.06 (p = 1.79e-09), whereas *the number of queries used*, *the success/-failure rates*, and *the order of the queries* did not influence the SUS ratings. It seems that the factor *time* is a very important issue for casual end-users when judging a user interface. The *order in which the interfaces were presented* to the user made an impact: The system that was tested last always obtained a higher SUS score (p = 0.0025), i.e., an increase by 5.3. *Knowledge of informatics* was the only additional variable that also influenced the SUS ratings: The better the knowledge of informatics of a user was, the higher the SUS score turned out for each interface (p = 0.0029).

	positive comments on the interface	negative comments on the interface
NLP-Reduce	+ the simplest interface (5) + similar to common search engines (2)	– bad presentation of results (5) – too relaxed (2)
Querix	+ simple to use (19) + free and unrestricted query language (7) + provides clear and good answers (4)	
Ginseng	+ simple (3) + comprehensible (2)	– too restrictive (4) – too complicated (3)
Semantic Crystal	+ graphical display of elements(5) + different (3) + seems reliable (2)	– too complicated (18) – too laborious (7) – not comprehensible (2)

Table 6.7: The comments most often provided by the test users for each query interface. The numbers in brackets indicate how often the comment was given.

When categorizing and counting the **comments** that users gave in the comparison questionnaires to describe the motivation for their choices of the best- and least-liked

interfaces, the most common responses for each interface were the ones presented in Table 6.7. The number of times a comment was given is indicated in parentheses.

Obviously, NLP-Reduce and Querix were deemed simple to use, but NLP-Reduce's answer presentation which includes complete URIs was disliked. Although the manner of Ginseng's use was easy to comprehend, it was considered to be too restrictive and, therefore, too complicated because the users were obliged to follow a fixed vocabulary and prescribed sentence structures. The comments for Semantic Crystal are controversial as expected: While some users liked the graphical display of what was possible to ask and were intrigued by the different approach, most subjects rated it too complicated and too time-consuming.

	positive comments on the query language	negative comments on the query language
NLP-Reduce	+ I can use keywords (5) + no thinking required (2) + robust to input (2)	– query language not clear (4) – no superlative forms (3)
Querix	+ I can use my language (8) + simple to use (5) + clear language (2)	– one has to enter complete sentences (2)
Ginseng	+ helpful (4) + simple(3)	– too restrictive (3)
Semantic Crystal	+ playful character (2)	– too laborious (7) – too complicated (5) – cumbersome (4)

Table 6.8: The comments most often provided by the test users for each specific query language. The numbers in brackets indicate how often the comment was given.

In addition, the comparison questionnaire specifically asked the test users for which query language they liked best and least and why. The comments that were given for each query language are listed in Table 6.8. Again, the number of times a comment was given is indicated in parentheses.

The comments most often given for the query languages of the four systems are consistently contradictory. While the use of keywords in NLP-Reduce was rated positively by some users, for others NLP-Reduce's query language was unclear. Most users liked the every-day and clear language of Querix, whereas some found it cumbersome that one must enter complete sentences. Ginseng's query language was considered both helpful as well as controlling. Finally, the graphical query composition language of Semantic Crystal was appealing to some users due its playful character, but most subjects clearly

expressed a distaste for the query language because it is too complicated and laborious.

The following comments were found striking enough that they have been listed individually:

1. NLP-Reduce is too formal.
2. The language is lost in NLP-Reduce.
3. It is not clear what language can be used in NLP-Reduce.
4. With NLP-Reduce, I can use normal speech patterns.
5. NLP-Reduce's language is too unrestricted to give confidence.
6. No structured queries in NLP-Reduce.
7. Querix has clear sentence structures. Its language is everyday language.
8. Semantic Crystal is fun, but too laborious for everyday use.
9. Semantic Crystal is more difficult to use than a system allowing sentences.
10. The language of Semantic Crystal is very natural.
11. Ginseng and Semantic Crystal appear innovative, but too restrictive. NLP-Reduce is too relaxed. Querix is a good compromise.

Noticeably, there are again conflicts in the comments with regard to which query language is regarded as formal and which as natural. Consider the first (no. 1) and the tenth (no. 10) comment, for example; they argue an arrangement of languages exactly the opposite of that with which we placed the query languages in the Formality Continuum. Furthermore, NLP-Reduce is highly controversial: while some declare its query language to be confusing (no. 2, 3, 5), others find it very natural (no. 4). We can count five comments (no. 2, 3, 5, 6, 11) asking for more structure in the query language of NLP-Reduce; these are prime examples of the habitability problem.

Comment no. 9 about Semantic Crystal ("fun, but too laborious for everyday use") raises the issue that the usefulness of a query interface may depend on how often the interface is used and, to carry the idea a bit further, for which tasks.

The structure that is imposed by Querix's language seems to be accepted by end-users (comments no. 7 and 11). They may experience the structure of natural language sentences as natural and flexible; they do not even perceive it as structure. However, a remarkable number of users do notice the structure and appreciate it as assistance, therefore supporting our habitability hypothesis.

Statement no. 11, "Ginseng and Semantic Crystal appear innovative, but too restrictive. NLP-Reduce is too relaxed. Querix is a good compromise" nicely summarizes the concept of our Formality Continuum.

6.3 Discussion of the Most Remarkable Results

The results of the usability study, with feedback from 48 users, clearly show that *Querix and its query language, which allows full English questions with fix sentence beginnings, was judged the most useful and best-liked query interface*. This finding contradicts another usability study investigating different query languages and showing that students generally preferred keyword-based search over full-question search [Reichert et al., 2005]. The users in that study declared that they would only accept full query sentences if the retrieval results were better. In contrast, our results exhibit a highly significant preference for full-sentence queries almost independent of the retrieval performance, although five of 48 subjects stated that they liked the use of keywords with NLP-Reduce.

One of the most prominent qualitative results was that several users, who rated Querix as the best interface, explicitly stated that they appreciated the "freedom of the query language." Nevertheless, full sentences are more restrictive than keywords meaning that the query language of NLP-Reduce actually offers more freedom and less control than Querix. Additionally, the sentence beginnings accepted by Querix are limited to a set of six sentence beginnings, which restricts the query language even more. We can think of two reasons for the comment:

1. With full-sentence questions, users can communicate their information need in a familiar and natural way without having to think of appropriate keywords in order to find what they are looking for.

2. People can express more semantics when they use full sentences and not just keywords. Using verbs and prepositions to link loosely listed nouns enables semantic associations, which users may experience as more freedom in query formulation.

The analysis of the results reveals a divergence between the perceived and the actual correctness of answers. Systems such as Querix, which generate natural language answers and engage users in some kind of natural language feedback or clarification dialogs, apparently lead to the impression that the interface "understands" the user, therefore creating confidence towards the returned answers. We think that this is one of the reasons that our subjects rated Querix best with regard to the SUS score as well as by why they directly named the system they liked best. Though NLP-Reduce exhibited a

better (but not significant) objective success retrieval performance, it was rated less favorably than Querix. Therefore, retrieval performance seems to not be the primary criterion that creates confidence towards an interface in general. Nevertheless, the preference for Querix and its full-sentence query language was extremely significant. As such, the result of the study is doubtlessly that Querix was the best-liked and best-rated query interface.

Although the success rate that was achieved by the subjects with Semantic Crystal was the lowest of the four interfaces, we actually think that this is a good result when considering that our subjects used the interface for the first time, that they were completely unfamiliar with ontology and SPARQL issues, and that "they were thrown in the deep end of query composing tasks" with Semantic Crystal after only very brief instructions—which were even given on paper and not by a live system demo. Furthermore, though Semantic Crystal was assessed as difficult and laborious to use, some users pointed out the big advantage of graphically displayed knowledge bases and queries. Consequently, we should consider interfaces to Semantic Web data that offer *a combination of graphically displayed as well as keyword-based and full-sentences query languages*. A user could then choose between different querying possibilities. And we might have to think of adequate *natural language answer generation components* [Androutsopoulos et al., 2005], which seems to increase a user's trust in a system and the overall user satisfaction.

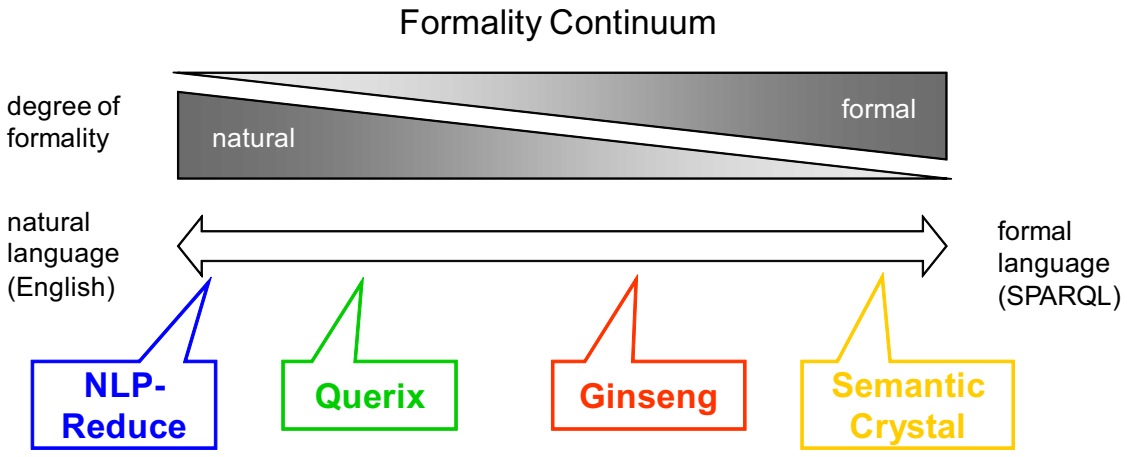


Figure 6.2: The four query interfaces NLP-Reduce, Querix, Ginseng, and Semantic Crystal to query Semantic Web data support different query languages with regard to their degree of freedom, naturalness, structuredness, and formality, therefore providing different positions distributed along the Formality Continuum.

To get back to the habitability hypothesis, we must first recall it: The habitability hypothesis proposes that query interfaces to the Semantic Web should impose some structure on the casual end-user to guide the query formulation process, but not overly control the user with an excessively formalistic language. As such, the best solutions for casual

or occasional end-users lie somewhere in the middle of the Formality Continuum (cf. Figure 6.2), therefore easing the query formulation task.

The usability study almost fully supports the hypothesis, since Querix, which requires the structure of full English sentences with a limited set of sentence beginnings significantly outperformed the other three interfaces when evaluated by 48 casual end-users. The structure that is imposed by Querix, however, is not perceived as a formal structure but as a natural, guiding structure, which is plausible because the structure of natural language is not noticed in everyday use either. On the other hand, the structure that was imposed by Ginseng was evidently too restrictive.

> Therefore, the best solutions for casual end-users lie towards the middle, but definitely on the natural side of the Formality Continuum.

This conclusion also agrees with the findings of the performance evaluation, which revealed that by slightly restricting the natural query language, we can increase the quality of the retrieval performance and, as such, offer a reliable as well as helpful query interface to the Semantic Web.

7

Limitations and Future Work

While the results from both the retrieval performance evaluation and the usability study are encouraging, many important challenges remain. The thesis provides a good basis for a deeper exploration and evaluation of NLIs to Semantic Web knowledge bases. The overarching goal is to turn the vision of the Semantic Web into reality, which can only happen if we bridge the gap between the real-world end-users and the formal logic underlying the Semantic Web. More specifically, we would like to further investigate our hypotheses regarding the Formality Continuum to the fullest extent in order to advance our scientific knowledge regarding NLIs for the Semantic Web by relying on natural language technology.

Although our three NLIs and the formal, graphical approach to query Semantic Web resources showed that low-complexity NLIs achieve high precision performance, and that end-users seem to favor full natural language search interfaces, even accepting some restrictions towards the questions' structures obtaining guidance in exchange, we can find various limitations concerning the interfaces we developed, the retrieval performance evaluation, and the usability study.

7.1 Interfaces

We will first improve each of our four prototype systems on the basis of what we have learned from our two evaluations in order to surmount each system's teething troubles. We are not aiming at improving recall, which would mean undermining the habitability hypothesis by extending the deliberately controlled query languages, but at increasing precision as well as the interaction usability in general.

NLP-Reduce clearly struck its limits with questions containing negations, comparatives, or superlatives. We will not implement negation in order to avoid complex semantic analysis and, hence, a conflict with the adaptability hypothesis. However, we will

find a straightforward and efficient way to process comparative and superlative forms of adjectives, as people apparently find them useful when expressing information needs. We urgently need to remove all URIs from the graphical user interface of NLP-Reduce, a feature that was exceedingly disliked by the test users taking part in our usability study. Furthermore, we will position the generated SPARQL queries less prominently and provide short instructions on the query language that can be used with NLP-Reduce. The latter issue is intended to establish confidence towards the answers returned by the interface, which are, in fact, reliable.

Lengthy and complex sentences depressed the precision of Querix. We will, therefore, generally investigate how we can further use information that is provided in a question's syntax tree after parsing. The goal is to employ syntactical relationships that exist in natural language questions in order to improve our triple matching algorithm. Additionally, since end-users found it very helpful, we intend to fully exploit the clarification dialog feature of Querix.

The results of our evaluations showed that Ginseng, while scoring modestly in the usability study overall, was assessed as a very useful tool by some subjects, and maintains very good retrieval performance. We, therefore, believe that the approach is suitable for certain occasional and casual end-users querying the Semantic Web. By extending and advancing the static grammar of Ginseng, we want to further investigate the potential of controlled, quasi-natural interfaces that neither require the learning of a formal query language nor any expert knowledge. In addition, we are currently working on an Italian grammar for Ginseng such that Italian knowledge bases can be loaded and queried. Preliminary results have shown that multilingualism can be straightforwardly realized in Ginseng [D'Onofrio, 2007].

Certainly, we will also improve Semantic Crystal, as the usability study revealed its usability and attractiveness even more than with Ginseng. Our plan is to give the graphical representation of an ontology model a different and improved layout. We think that the moving graph elements may be enjoyable when using the interface for the first time, but that they might get tiresome after a while. Moreover, we intend to incorporate fisheye technologies for graphs, thereby facilitating the display of large ontology model graphs.

Consolidating the results of the usability study, we will, moreover, design and implement one more query interface for casual end-users that offers a combination of graphically displayed and natural language query languages which embeds both keywords as well as full sentences. A user can then choose which query language to use according to personal preferences or different information seeking tasks. Motivated by the work of Wang and Parsia, who demonstrate and evaluate the advantage of graphically displayed ontologies [Wang and Parsia, 2006], we assume that visualized ontologies are one way of letting users know what is possible to ask and, therefore, is particularly useful for casual end-users. The need for both keywords and full sentences are supported by the

findings of Dittenbach *et al.*, who discovered that end-users enter full sentences, sentence fragments, and keywords if they are free to choose [Dittenbach et al., 2003]. The new interface would have to be thoroughly evaluated in a retrieval performance as well as usability evaluation.

Furthermore, we might have to develop adequate *natural language answer generation components* [Androutsopoulos et al., 2005, Minock, 2005]. Natural language answers such as "The height of Whistler Mountain is 2181 meters." would presumably better suit casual end-users than a single number or a table containing an answer set. Besides, such answers seem to increase a user's confidence towards a system and the overall user satisfaction.

7.2 Evaluations

The retrieval performance evaluation conducted in the course of this thesis in order to test the adaptivity hypothesis was limited with regard to the number of test sets and domains. Furthermore, the data sets comprised rather small ontology models, i.e., the number of classes and properties. As such, the controlled benchmark evaluation did not provide insights towards the performance of our three NLIs in a real-world setting. We, therefore, intend to perform a thorough evaluation of the interfaces and the new combined interface with real data in real Web interface settings. We can think of data sets such as the highly frequented web sites "`hitparade.ch`," "`wsl.ch`", the site of the Swiss Federal Institute for Forest, Snow and Landscape Research, and "`tourismus-schweiz.ch`" are built upon. We would export and transform the databases into a suitable format. The Web interfaces would allow us to collect a series of real English questions. Then, we could hand-code the correct answers or provide Web site users with the opportunity to give feedback to returned answers indicating whether they were satisfied with an answer not. The feedback mechanism would have to be easy and convenient to encourage users to give feedback. In order to increase the external validity, we hope that these databases offer a more generalizable set of questions than the Mooney Natural Language Learning Data.

We are well aware that our usability study does not provide a definitive answer to the general discussion of the usefulness of NLIs. Concerning valid conclusions to be drawn from a usability study, we would still need a more comprehensive usability study with more users in order to cover more precisely the distinguished degrees of query languages along the Formality Continuum. To prevent influences from variables that are not directly linked to the query languages, the NLIs should be the same except for the query languages. In our study, the appearance of the interfaces was different.

We limited ourselves to four interfaces and four queries for several reasons. Firstly,

we wanted to cover each possible tool order; consider that a usability study with five different interfaces requires 120 users to cover each order of the interfaces. Second, we preferred to not overload the users in an exhaustive experiment that would, due to fatigue, risk tainting the results. Lastly, our users should not be students (most usability studies work with students [Lee et al., 2006, Joachims et al., 2007, Reichert et al., 2005]), but people representing a general public. Finding such users is a difficult, time-consuming, and also expensive endeavor, since we offered our users a small monetary reward for taking part.

We still believe that our usability study provides a substantial contribution to the discussion of how useful NLIs are for casual end-users. Together with the conclusions from the retrieval performance evaluation, we hope to have established a basis for further research addressing the issue of natural language query interfaces to the Semantic Web for casual end-users in the future.

8

Conclusions

The vision of the Semantic Web is the development of a distributed, organically growing ontology-based knowledge base that is machine-processable and can be accessed and extended by all users in a manner akin to the traditional WWW. To that end, the Semantic Web is being designed on top of formal logic frameworks. Casual users, however, feel very uncomfortable about using it, or even unable to command the formal logic concepts underlying the Semantic Web. So how can we bridge this obvious gap between the formal logic-based Semantic Web and the casual end-users? One solution for addressing the gap is the use of NLIs that allow users to access information repositories by using familiar natural language. However, most current NLP approaches require computationally intensive algorithms and utilize vast quantities of background knowledge, which results in highly domain-dependent tools without providing evidence that natural language query languages are in fact appropriate for casual end-users.

The overarching goal of this thesis was to turn the vision of the Semantic Web a little further into realization by bridging the gap between the real-world users and the logic-based underpinnings of the Semantic Web, which can only happen if we open its capabilities to the general public. As such, the thesis proposed to break the dichotomy between full natural language approaches and formal, logic-based query approaches regarding them as ends of a Formality Continuum, where the freedom of full natural languages and the structuredness of formal query languages lie at the ends of the continuum. We hypothesized that we can overcome the portability problem of typical NLIs lying at the natural end of the Formality Continuum without having to apply complex, knowledge-intensive algorithms and undertake time-consuming implementation efforts by extracting the necessary knowledge needed to process natural language queries from semantically-enriched knowledge bases and controlling the query language. We called this the *adaptivity hypothesis*. Furthermore, we hypothesized that natural language query interfaces offer a real alternative for casual end-users to interact with the Semantic Web and its logic-based knowledge bases—assuming that we can address the problem of NLIs

only being used successfully if users know what is possible to ask by using a natural query language that is guided and controlled to some extent in order to support the user in query formulation tasks. As such, this *habitability hypothesis* proposed that the best interaction approach for the casual and occasional Semantic Web user lies towards the middle of the Formality Continuum.

To support our propositions, we presented four different query interfaces to the Semantic Web, all of which lie at different positions of the Formality Continuum: NLP-Reduce, Querix, Ginseng, and Semantic Crystal. The first two interfaces allow users to pose question in full or slightly controlled English. The third interface offers query formulation in a controlled language akin to English. The last interface belongs to the formal approaches, as it exhibits a formal, but graphically displayed query language. We intended to forgo the need for full natural language processing machinery, avoiding all the computational and linguistic complexities involved with such an endeavor. Furthermore, we could bypass a prohibitive formal query language that can only be reasonably managed by experts while still offering rich tools for the composition of complex queries by casual users.

With three of our four different query interfaces, NLP-Reduce, Querix, and Ginseng, we conducted a thorough test set evaluation showing the retrieval performance of the interfaces in terms of recall and precision with three data sets from the domains geography, restaurants, and jobs. We applied 1748 natural language test questions. Since the test sets were used in previous test studies, they enabled a comparison of our NLIs with the three existing natural language systems COCKTAIL [Tang and Mooney, 2001], PANTO [Wang et al., 2007], and PRECISE [Popescu et al., 2003]. This evaluation allowed us to generate conclusive evidence regarding the adaptivity hypothesis. We found that our NLIs achieved moderate recall values due to the restrictedness of their query languages, but very high precision performance values, which roughly increased with ascending degree of the structuredness of the query languages. The adaptivity hypothesis has largely been corroborated, confirming that portable, slightly controlled NLIs to the Semantic Web with low complexity and with a high quality of retrieval performance can be developed.

When mapping the retrieval performance of the three interfaces, NLP-Reduce, Querix, and Ginseng, to their increasing degree of formality along the Formality Continuum, the mapping in terms of precision was semi-successful, since the average precision achieved by the least controlling system, NLP-Reduce, was better than the second least restrictive system, Querix. However, the controlled query language of Ginseng performed as expected. On the other hand, the declining recall values from NLP-Reduce to Querix and Ginseng inversely reflect the increasing degree of formality and control of the systems' query languages. As such, the overall concept of the Formality Continuum was confirmed.

Similar to our interfaces, most modern approaches make use of off-the-shelve tools and build on techniques that are shallow from an NLP perspective in order to achieve robustness, portability, and high-quality retrieval performance. The progress is due to the great quantity of semantic information that is enclosed as classes and relationships in ontologies and ontology-based knowledge bases, and as such offers a rich platform in the query analysis and translation process—whereas the semantic information of attributes and values in databases seems to be limited. Our approaches further tried to exploit the reduction of complexity and the technical setup to a captivating simplicity.

In a second evaluation step, we conducted a comprehensive usability study by benchmarking all four tools against each other in a controlled experiment in order to test the habitability hypothesis. The goal was that casual end-users tested and assessed the usability of each of our four systems and, in particular, their different query languages. The study with 48 users from the street revealed that the full sentence query option was significantly preferred to keywords as well as the menu-guided and the graphical query languages. Hence, the habitability hypothesis was also largely supported, showing that, as structuration theory would predict [Giddens, 1984, Orlikowski, 1992], casual end-users, in order to be assisted in query formulation tasks, favor query languages that impose some structure but do not overly restrict them. NLIs lying towards the natural middle of the Formality Continuum, therefore, offer adequate query languages and can be considered to be useful for casual end-users.

While the results of the system usability score clearly show that Querix and its query language, which allow full English questions with a limited set of sentence beginnings, were, with incredible significance, the most useful and best-liked query interface for users, the analysis of the results reveal a divergence between the perceived and the actual restrictedness and structure of a query language. The test users thought that the query language of Querix allows more freedom than the keyword-based approach. However, full sentences are more restrictive than keywords; the habitability problem occurred in its purest form. Casual and occasional information seekers seem to be willing to enter more than just keywords because they can express more semantics in full sentences and be more specific. On the other hand, their sentences are usually of a rather simple syntactical manner, and, moreover, do not combine many concepts into one question. This confirms that complex and tedious full NLP scaffolds are not necessary to avoid formal query language interfaces.

The usability study's outcome caused us to contemplate developing an interface for casual end-users that offers a combination of graphically displayed and natural language query languages which embeds both keywords as well as full sentences. A user can then choose which query language to use according to personal preferences or different information seeking tasks. A thorough evaluation of this new interface in a real-world setting would demonstrate its usefulness. Realizing these ideas will be our future undertaking.

Though the results from the two evaluations supporting our propositions were encouraging, many important challenges remain. As such, our thesis provides a good basis for a deeper exploration and evaluation of NLIs to Semantic Web knowledge bases, and a substantial contribution to future discussions of how well they perform and how useful NLIs are for casual end-users. We hope that this thesis inspires and encourages other researches in the intermixed area of Semantic Web and NLI development to build natural language query interfaces and, particularly, to carry out usability studies. We are indeed aware that usability studies are demanding and time-consuming endeavors that have to be planned and conducted extremely carefully. The reward, however, is invaluable for both the end-users as well as the query interfaces.

The need to make the contents of the Semantic Web accessible to end-users has become increasingly pressing as the amount of information stored in ontology-based knowledge bases steadily increases. People's familiarity with natural language might be the key to simplify their interaction with ontologies and offer the capabilities of the Semantic Web to the general public.

References

[Andreasen, 2003] Andreasen, T. (2003). An Approach to Knowledge-based Query Evaluation. *Fuzzy Sets and Systems*, 140(1):75–91.

[Androutsopoulos et al., 2005] Androutsopoulos, I., Kallonis, S., and Karkaletsis, V. (2005). Exploiting OWL Ontologies in the Multilingual Generation of Object Descriptions. In *10th European Workshop on Natural Language Generation (ENLG 2005)*, pages 150–155, Aberdeen, UK.

[Androutsopoulos et al., 1995] Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. (1995). Natural Language Interfaces to Databases - An Introduction. *Natural Language Engineering*, 1(1):29–81.

[Antoniou and Hermelen, 2004] Antoniou, G. and Hermelen, F. v. (2004). *A Semantic Web Primer*. MIT Press, Cambridge, MA.

[Auer and Lehmann, 2007] Auer, S. and Lehmann, J. (2007). What have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content. In *4th European Semantic Web Conference (ESWC 2007)*, pages 503–517, Innsbruck, Austria.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK.

[Badia, 2007] Badia, A. (2007). Question Answering and Database Querying: Bridging the Gap with Generalized Quantification. *Journal of Applied Logic*, 5(1):3–19.

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman, Harlow, UK.

[Baker et al., 1998] Baker, P. G., Brass, A., Bechhofer, S., Goble, C., Paton, N., and Stevens, R. (1998). TAMBIS: Transparent Access to Multiple Bioinformatics Informa-

tion Sources. An Overview. In *6th International Conference on Intelligent Systems for Molecular Biology (ISMB-98)*, pages 25–34, Montreal, Canada.

[Battle, 2006] Battle, L. (2006). Preliminary Inventory of Users and Tasks for the Semantic Web. In *3rd International Semantic Web User Interaction Workshop (SWUI 2006)*, Athens, GA.

[Bechhofer and Goble, 1999] Bechhofer, S. and Goble, C. (1999). Classification Based Navigation for Picture Archives. In *IFIP WG2.6 Eighth Working Conference on Data Semantics (DS8)*, pages 291–310, Rotorua, New Zealand.

[Bechhofer and Ng, 2006] Bechhofer, S. and Ng, G. (2006). OilEd. `http://oiled.man.ac.uk/`.

[Bechhofer et al., 1999] Bechhofer, S., Stevens, R., Ng, G., Jacoby, A., and Goble, C. (1999). Guiding the User: An Ontology Driven Interface. In *1999 User Interfaces to Data Intensive Systems (UIDIS 1999)*, pages 158–161, Edinburgh, Scotland.

[Beckett, 2004] Beckett, D. (2004). RDF/XML Syntax Specification (Revised). W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`.

[Bederson, 2000] Bederson, B. B. (2000). Fisheye Menus. In *ACM UIST Symposium on User Interface Software and Technology*, pages 217–225, San Diego, CA.

[Bell and Rowe, 1992] Bell, J. E. and Rowe, L. A. (1992). An Exploratory Study of Ad Hoc Query Languages to Databases. In *8th International Conference on Data Engineering*, pages 606–613, Tempe, AZ.

[Berners-Lee, 2006] Berners-Lee, T. (2006). Notation 3. Last change on March 9, 2006. `http://www.w3.org/DesignIssues/Notation3`.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific America*, 284(5):34–43.

[Bernstein and Kaufmann, 2006] Bernstein, A. and Kaufmann, E. (2006). GINO - A Guided Input Natural Language Ontology Editor. In *5th International Semantic Web Conference (ISWC 2006)*, pages 144–157.

[Bernstein et al., 2004] Bernstein, A., Kaufmann, E., Bürki, C., and Klein, M. (2004). Object Similarity in Ontologies: A Foundation for Business Intelligence Systems and High-performance Retrieval. In *Twenty-Fifth International Conference on Information Systems (ICIS 2004)*, pages 11–25.

[Bernstein et al., 2005a] Bernstein, A., Kaufmann, E., Göhring, A., and Kiefer, C. (2005a). Querying Ontologies: A Controlled English Interface for End-users. In *4th International Semantic Web Conference (ISWC 2005)*, pages 112–126.

[Bernstein et al., 2005b] Bernstein, A., Kaufmann, E., and Kaiser, C. (2005b). Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine. In *15th Workshop on Information Technologies and Systems (WITS 2005)*, pages 112–126, Las Vegas, NV.

[Bernstein et al., 2006] Bernstein, A., Kaufmann, E., Kaiser, C., and Kiefer, C. (2006). Ginseng: A guided input natural language search engine for querying ontologies. In *2006 Jena User Conference*, Bristol, UK.

[Blau et al., 2002] Blau, H., Immerman, N., and Jensen, D. (2002). A Visual Language for Querying and Updating Graphs. Technical Report 2002-037, University of Massachusetts, Amherst, MA.

[Bortz and Döring, 2002] Bortz, J. and Döring, N. (2002). *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. Springer, Berlin.

[Bowen et al., 2004] Bowen, P. L., Chang, C.-J. A., and Rohde, F. H. (2004). Non-Length Based Query Challenges: An Initial Taxonomy. In *Fourteenth Annual Workshop on Information Technologies and Systems (WITS 2004)*, pages 74–79, Washington, D.C.

[Breitman et al., 2007] Breitman, K. K., Casanova, M. A., and Truszkowski, W. (2007). *The Semantic Web. Concepts, Technologies and Applications*. Springer, London.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/rdf-schema/`.

[Brooke, 1996] Brooke, J. (1996). SUS - A "quick and dirty" Usability Scale. In Jordan, P., Thomas, B., Weerdmeester, B., and McClelland, A., editors, *Usability Evaluation in Industry*. Taylor and Francis, London.

[Buckland and Gey, 1999] Buckland, M. and Gey, F. (1999). The Relationship between Recall and Precision. *Journal of the American Society for Information Science*, 45(1):12–19.

[Cha, 1991] Cha, S. K. (1991). Kaleidoscope: A Cooperative Menu-guided Query Interface (SQL Version). *IEEE Transactions on Knowledge and Data Engineering*, 3(1):42–47.

[Chakrabarti, 2004] Chakrabarti, S. (2004). Breaking through the Syntax Barrier: Searching with Entities and Relations. In *15th European Conference on Machine Learning (ECML 2004)*, pages 9–16, Pisa, Italy.

[Charniak, 2000] Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *North American Chapter of the Association for Computational Linguistics*, pages 132–139, New Brunswick, NJ.

[Cimiano, 2004] Cimiano, P. (2004). ORAKEL: A Natural Language Interface to an F-Logic Knowledge Base. In *9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004)*, pages 401–406, Salford, UK.

[Cimiano et al., 2007] Cimiano, P., Haase, P., Heizmann, J., and Mantel, M. (2007). ORAKEL: A Portable Natural Language Interface to Knowledge Bases. Technical Report, Institute AIFB, University of Karlsruhe, Germany.

[Clark & Parsia LLC, 2007] Clark & Parsia LLC (2007). Pellet. Last version 1.5.0 released on July 20, 2007. `http://pellet.owldl.com/`.

[Croson, 2005] Croson, R. (2005). The Method of Experimental Economics. *International Negotiation*, 10(1):131–148.

[Cunningham et al., 2007] Cunningham, H., Bontcheva, K., Tablan, V., and Maynard, D. (2007). GATE - General Architecture for Text Engineering. Last version 4.0 released in July 2007. `http://gate.ac.uk/`.

[Davies et al., 2002] Davies, J., Fensel, D., and Harmelen, F. v. (2002). *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley & Sons, Inc., New York, NY.

[Dekleva, 1994] Dekleva, S. M. (1994). Is Natural Language Querying Practical? *ACM SIGMIS Database*, 25(2):24–36.

[Désert, 1993] Désert, S. E. (1993). WESTLAW Is Natural v. Boolean Searching: A Performance Study. *Law Library Journal*, 85(4):713–742.

[Distelhorst et al., 2003] Distelhorst, G., Srivastava, V., Rosse, C., and Brinkley, J. F. (2003). A Prototype Natural Language Interface to a Large Complex Knowledge Base, the Foundational Model of Anatomy. In *American Medical Informatics Association Annual Fall Symposium*, pages 200–204, Philadelphia, PA.

[Dittenbach et al., 2003] Dittenbach, M., Merkl, D., and Berger, H. (2003). A Natural Language Query Interface for Tourism Information. In *10th International Conference on Information Technologies in Tourism (ENTER 2003)*, pages 152–162, Helsinki, Finland.

[D'Onofrio, 2007] D'Onofrio, U. (2007). Ginseng Goes Italian: Adding Multilingualism to a Natural Language Search Engine. Diploma Thesis, University of Zurich, Switzerland.

[Dublin Core Medatata Initiative, 2007] Dublin Core Medatata Initiative (2007). Last updated on August 20, 2007. `http://dublincore.org/`.

[Duke et al., 2007] Duke, A., Glover, T., and Davies, J. (2007). Squirrel: An Advanced Semantic Search and Browse Facility. In *4th European Semantic Web Conference*, Innsbruck, Austria.

[Eyeball, 2007] Eyeball (2007). Checking RDF/OWL for Common Problems. Last visited on August 30, 2007. `http://jena.sourceforge.net/Eyeball/`.

[Faraway, 2005] Faraway, J. J. (2005). *Linear Models with R*. Chapman & Hall, Boca Raton, FL.

[Faraway, 2006] Faraway, J. J. (2006). *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models*. Chapman & Hall, Boca Raton, FL.

[Feldman, 1996] Feldman, S. (1996). Testing Natural Language. Comparing Dialog, Target, and DR-LINK. *Online*, 20(6):71–79.

[Fischer, 2006] Fischer, L. (2006). NLP-Reduce - Ein natürlichsprachliches Suchsystem für "Semantic Web" Daten. Diploma Thesis, University of Zurich, Switzerland.

[Frank et al., 2007] Frank, A., Krieger, H.-U., Xu, F., Uszkoreit, H., Crysmann, B., Jrg, B., and Schäfer, U. (2007). Question Answering from Structured Knowledge Sources. *Journal of Applied Logic*, 5(1):20–48.

[Giddens, 1984] Giddens, A. (1984). *The Constitution of Society: Outline of the Theory of Structuration*. University of California Press, Berkeley, CA.

[Grant and Beckett, 2004] Grant, J. and Beckett, D. (2004). RDF Test Cases. W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/rdf-testcases/`.

[Grosz et al., 1987] Grosz, B. J., Appelt, D. E., Martin, P. A., and Pereira, F. C. (1987). TEAM: An Experiment in the Design of Transportable Natural Language Interfaces. *Artificial Intelligence*, 32(2):173–243.

[Gruber, 1992] Gruber, T. R. (1992). Ontolingua: A Mechanism to Support Portable Ontologies. Technical Report KSL-91-66, Knowledge Systems Laboratory, Stanford University, CA.

[Guarino et al., 1999] Guarino, N., Masolo, C., and Vetere, G. (1999). OntoSeek: Content-based Access to the Web. *IEEE Intelligent Systems*, 14(3):70–80.

[Hallett et al., 2005] Hallett, C., Power, R., and Scott, D. (2005). Intuitive Querying of e-Health Data Repositories. In *UK E-Science All-hands Meeting*, Nottingham, UK.

[Hayes, 2004] Hayes, P. (2004). RDF Semantics. W3C Recommendation, February 10, 2004.

[Heer, 2007] Heer, J. (2007). The Prefuse Visualization Toolkit. A Java-based Toolkit for Building Interactive Information Visualization Applications. Last modified on August 23, 2007. `http://www.prefuse.org`.

[Hempelmann et al., 2005] Hempelmann, C. F., Rus, V., Graesser, A. C., and McNamara, D. S. (2005). Evaluating State-of-the-Art Treebank-style Parsers for Coh-Metrix and Other Learning Technology Environments. In *2nd Workshop on Building Educational Applications Using NLP*, pages 69–76, Ann Arbor, MI.

[Hersh et al., 1999] Hersh, W., Price, S., Chan, B., Kraemer, D. K., Sacherek, L., and Olson, D. (1999). System and User Attributes Associated with Successful Searching. In *IEEE Advances in Digital Libraries Conference*, pages 60–70, Baltimore, MD.

[Jarke et al., 1985] Jarke, M., Turner, J. A., Stohr, E. A., Vassiliou, Y., White, N. H., and Michielsen, K. (1985). A Field Evaluation of Natural Language for Data Retrieval. *IEEE Transactions on Software Engineering*, (1):97–113.

[Jena, 2007] Jena (2007). A Semantic Web Framework for Java. Last visited on August 30, 2007. `http://jena.sourceforge.net/`.

[Joachims et al., 2007] Joachims, T., Granka, L., Pan, B., Hembrooke, H., Radlinski, F., and Gay, G. (2007). Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search. *ACM Transactions on Informations Systems*, 25(2):1–27.

[Joseki, 2007] Joseki (2007). A SPARQL Server for Jena. Last visited on August 30, 2007. `http://www.joseki.org/`.

[Kaiser, 2004] Kaiser, C. (2004). Ginseng - A Natural Language User Interface for Semantic Web Search. Diploma Thesis, University of Zurich, Switzerland.

[KAON2, 2007] KAON2 (2007). Last version 2 released on August 1, 2007. `http://kaon2.semanticWeb.org`.

[Katz et al., 2002] Katz, B., Lin, J., and Quan, D. (2002). Natural Language Annotations for the Semantic Web. In *International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2002)*, Irvine, CA.

[Kaufmann and Bernstein, 2007] Kaufmann, E. and Bernstein, A. (2007). How Useful are Natural Language Interfaces to the Semantic Web for Casual End-users? In *6th International Semantic Web Conference (ISWC 2007)*, pages 281–294, Busan, Korea.

[Kaufmann et al., 2007] Kaufmann, E., Bernstein, A., and Fischer, L. (2007). NLP-Reduce: A "naïve" but Domain-independent Natural Language Interface for Querying Ontologies. In *4th European Semantic Web Conference (ESWC 2007)*, Innsbruck, Austria.

[Kaufmann et al., 2006] Kaufmann, E., Bernstein, A., and Zumstein, R. (2006). Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. In *5th International Semantic Web Conference (ISWC 2006)*, pages 980–981, Athens, GA.

[Klein and Manning, 2002] Klein, D. and Manning, C. D. (2002). Fast Exact Inference with a Factored Model for Natural Language Parsing. In *Advances in Neural Information Processing Systems (NIPS 2002)*, pages 3–10. `http://nlp.stanford.edu/software/lex-parser.shtml`.

[Klyne and Carroll, 2004] Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/rdf-concepts/`.

[Krivov, 2007] Krivov, S. (2007). GrOWL. Last visited on August 30, 2007. `http://www.uvm.edu/~skrivov/growl/`.

[Lee et al., 2006] Lee, H., Smeaton, A. F., and O'Connor, N. E. (2006). User Evaluation of Físchlár-News: An Automatic Broadcast News Delivery System. *ACM Transactions on Information Systems*, 24(2):145–189.

[Linckels and Meinel, 2005] Linckels, S. and Meinel, C. (2005). A Simple Solution for an Intelligent Librarian System. In *IADIS International Conference of Applied Computing (AC2005)*, pages 495–503, Lisbon, Portugal.

[Linckels and Meinel, 2006] Linckels, S. and Meinel, C. (2006). Resolving Ambiguities in the Semantic Interpretation of Natural Language Questions. In *Intelligent Data Engineering and Automated Learning (IDEAL 2006)*, pages 612–619, Burgos, Spain.

[Lopez et al., 2006a] Lopez, V., Motta, E., and Uren, V. (2006a). PowerAqua: Fishing the Semantic Web. In *3rd European Semantic Web Conference*, pages 393–410, Budva, Montenegro.

[Lopez et al., 2005] Lopez, V., Pasin, M., and Motta, E. (2005). AquaLog: An Ontology-portable Question Answering System for the Semantic Web. In *2nd European Semantic Web Conference (ESWC 2005)*, pages 546–562, Heraklion, Greece.

[Lopez et al., 2006b] Lopez, V., Sabou, M., and Motta, E. (2006b). PowerMap: Mapping the Real Semantic Web on the Fly. In *5th International Semantic Web Conference (ISWC 2006)*, pages 414–427, Athens, GA.

[Malhotra, 1975] Malhotra, A. (1975). *Design Criteria for a Knowledge-based English Language System for Management: An Experimental Analysis*. Ph.D. Thesis, MIT Sloan School of Management, Cambridge, MA.

[Manola and Miller, 2004] Manola, F. and Miller, E. (2004). RDF Primer. W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/rdf-primer/`.

[McCool, 2005] McCool, R. (2005). Rethinking the Semantic Web, Part 1. *Internet Computing, IEEE*, 9(6):88–87.

[McGuinness and Harmelen, 2004] McGuinness, D. L. and Harmelen, F. v. (2004). OWL Web Ontology Language Overview. W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/owl-features/`.

[Miller et al., 1993] Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). Introduction to WordNet: An On-line Lexical Database. Technical Report, Cognitive Science Laboratory, Princeton University, NJ.

[Minock, 2005] Minock, M. J. (2005). A Phrasal Approach to Natural Language Interfaces over Databases. Technical Report UMINF-05.09, University of Umea, Sweden.

[Minock, 2007] Minock, M. J. (2007). A STEP towards Realizing Codd's Vision of Rendezvous with the Casual User. In *33rd International Conference on Very large Data Bases (VLDB '07)*, pages 1358–1361, Vienna, Austria.

[MKBEEM, 2002] MKBEEM (2002). Multilingual Knowledge Based European Electronic Market Place. Last modification on April 23, 2002. `http://mkbeem.elibel.tm.fr/`.

[Mooney, 2004] Mooney, R. J. (2004). Learning Semantic Parsers: An Important but Under-studied Problem. In *AAAI 2004 Spring Symposium on Language Learning: An Interdisciplinary Perspective*, pages 39–44, Stanford, CA.

[Nielsen, 1993] Nielsen, J. (1993). *Usability Engineering*. Academic Press, San Diego/New York.

[Odgen and Bernick, 1997] Odgen, W. C. and Bernick, P. (1997). Using Natural Language Interfaces. In Helander, M., Landauer, T. K., and Prabhu, P. V., editors, *Handbook of Human Computer Interaction*. Elsevier, North-Holland.

[Orlikowski, 1992] Orlikowski, W. J. (1992). The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science*, 3(3):398–427.

[Paris and Tibbo, 1998] Paris, L. A. H. and Tibbo, H. R. (1998). Freestyle vs. Boolean: A Comparison of Partial and Exact Match Retrieval Systems. *Information Processing & Management*, 34(2):175–190.

[Popescu et al., 2003] Popescu, A.-M., Etzioni, O., and Kautz, H. (2003). Towards a Theory of Natural Language Interfaces to Databases. In *8th International Conference on Intelligent User Interfaces*, pages 149–157, Miami, FL.

[Porter, 1980] Porter, M. F. (1980). An Algorithm for Suffix Stripping. *Program*, 14(3):130–137. `http://tartarus.org/~martin/PorterStemmer/`.

[Preece et al., 2002] Preece, J., Rogers, Y., and Sharp, H. (2002). *Interaction Design: Beyond Human-Computer Interaction*. John Wiley and Sons, New York, NY.

[Prud'hommeaux and Seaborne, 2007] Prud'hommeaux, E. and Seaborne, A. (2007). SPARQL Query Language for RDF. W3C Candidate Recommendation, June 14, 2007. `http://www.w3.org/TR/rdf-sparql-query/`.

[Racer Systems, 2005] Racer Systems (2005). RacerPro 1.9. Last version 1.9.0 released on December 5, 2005. `http://www.racer-systems.com/`.

[Rasch et al., 2004] Rasch, B., Friese, M., Hofmann, W., and Naumann, E. (2004). *Quantitative Methoden - Band 1 und 2*. Springer, Berlin/Heidelberg, Germany.

[Rauterberg, 1991] Rauterberg, M. (1991). Benutzungs-orientierte Benchmark-Tests: eine Methode zur Benutzerbeteiligung bei Standardsoftware-Entwicklungen. In *Software Ergonomie'91 (German Chapter of ACM Berichte Nr. 32)*, pages 96–107, Stuttgart, Germany.

[Reichert et al., 2005] Reichert, M., Linckels, S., Meinel, C., and Engel, T. (2005). Student's Perception of a Semantic Search Engine. In *IADIS Cognition and Exploratory Learning in Digital Age (CELDA 2005)*, pages 139–147, Porto, Portugal.

[Reif, 2005] Reif, G. (2005). *WEESA - Web Engineering for Semantic Web Applications*. Dissertation, Vienna University of Technology, Faculty of Informatics, Vienna, Austria.

[RSS Advisory Board, 2007] RSS Advisory Board (2007). Really Simple Syndication Specifications, Tutorials and Discussion. Last version 2.0.9 published on June 6, 2007. http://www.rssboard.org/.

[Sachs, 2004] Sachs, L. (2004). *Angewandte Statistik: Anwendung statistischer Methoden*. Springer, Berlin.

[Salton and McGill, 1983] Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY.

[Schwitter and Tilbrook, 2004] Schwitter, R. and Tilbrook, M. (2004). Dynamic Semantics at Work. In *International Workshop on Logic and Engineering of Natural Language Semantics*, Kanazawa, Japan.

[Sesame, 2007] Sesame (2007). ... home of Sesame. Last visited on August 30, 2007. http://www.openrdf.org/.

[Sirin et al., 2007] Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53.

[Smith et al., 2004] Smith, M. K., Welty, C., and McGuinness, D. L. (2004). OWL Web Ontology Language Overview. W3C Recommendation, February 10, 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[Spink et al., 2001] Spink, A., Dietmar, W., Jansen, B. J., and Saracevic, T. (2001). Searching the Web: The Public and Their Queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234.

[Spoerri, 1993] Spoerri, A. (1993). InfoCrystal: A Visual Tool for Information Retrieval Management. In *Second International Conference on Information and Knowledge Management*, pages 11–20, Washington, D.C.

[Sprenger, 2006] Sprenger, B. (2006). Semantic Crystal - Ein End-User-Interface zur Unterstützung von Ontologie-Abfragen mit SPARQL. Diploma Thesis, University of Zurich, Switzerland.

[Staab and Studer, 2004] Staab, S. and Studer, R. (2004). *Handbook of Ontologies*. Springer, Berlin/Heidelberg/New York.

[Stanford Medical Informatics, 2007] Stanford Medical Informatics (2007). Protégé. Last visited on August 30, 2007. http://protege.stanford.edu/.

[Tablan et al., 2005] Tablan, V., Polajnar, T., Cunningham, H., and Bontcheva, K. (2005). User-friendly Ontology Authoring Using a Controlled Language. Research Memorandum CS-05-10, Department of Computer Science, University of Sheffield, UK.

[Tang and Mooney, 2001] Tang, L. R. and Mooney, R. J. (2001). Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. In *12th European Conference on Machine Learning (ECML-2001)*, pages 466–477, Freiburg, Germany.

[Team, 2007] Team, G. (2007). The GraphML File Format. Last updated on April 5, 2007. `http://graphml.graphdrawing.org/`.

[Tennant et al., 1983] Tennant, H. R., Ross, K. M., and Thompson, C. W. (1983). Usable Natural Language Interfaces through Menu-based Natural Language Understanding. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 154–160, Boston, MA.

[The Friend of a Friend Project, 2007] The Friend of a Friend Project (2007). FOAF. Last visited on August 30, 2007. `http://www.foaf-project.org/`.

[Thompson et al., 2005] Thompson, C. W., Pazandak, P., and Tennant, H. R. (2005). Talk to Your Semantic Web. *IEEE Internet Computing*, 9(6):75–78.

[Tomaiuolo and Packer, 1998] Tomaiuolo, N. G. and Packer, J. (1998). Maximizing Relevant Retrieval: Keyword and Natural Language Searching. *Online*, 22(6):57– 60.

[Tsarkov, 2007] Tsarkov, D. (2007). FaCT++. Last version 1.1.8 released on July 12, 2007. `http://owl.man.ac.uk/factplusplus/`.

[Turtle, 1994] Turtle, H. (1994). Natural Language vs. Boolean Query Evaluation: A Comparison of Retrieval Performance. In *17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '94)*, pages 212–220, Dublin, Ireland.

[W3C World Wide Web Consortium, 2007] W3C World Wide Web Consortium (2007). Last visited on August 30, 2007. `http://www.w3.org/`.

[Wang et al., 2007] Wang, C., Xiong, M., Zhou, Q., and Yu, Y. (2007). PANTO - A Portable Natural Language Interface to Ontologies. In *4th European Semantic Web Conference (ESWC 2007)*, pages 473–487, Innsbruck, Austria.

[Wang and Parsia, 2006] Wang, T. D. and Parsia, B. (2006). CropCircles: Topology Sensitive Visualization of OWL Class Hierarchies. In *5th International Semantic Web Conference (ISWC 2006)*, pages 695–708, Athens, GA.

[Witten and Eibe, 2005] Witten, I. H. and Eibe, F. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, Amsterdam, The Netherlands.

[Zumstein, 2006] Zumstein, R. (2006). Querix - A Natural Language Interface to Semantic Web Data. Diploma Thesis, University of Zurich, Switzerland.

List of Figures

List of Tables

A

Appendix

The ontology models of the three OWL knowledge bases from the domains of geography (Figure A.1), restaurants (Figure A.3), and job postings (Figure A.2), each represented as graph.

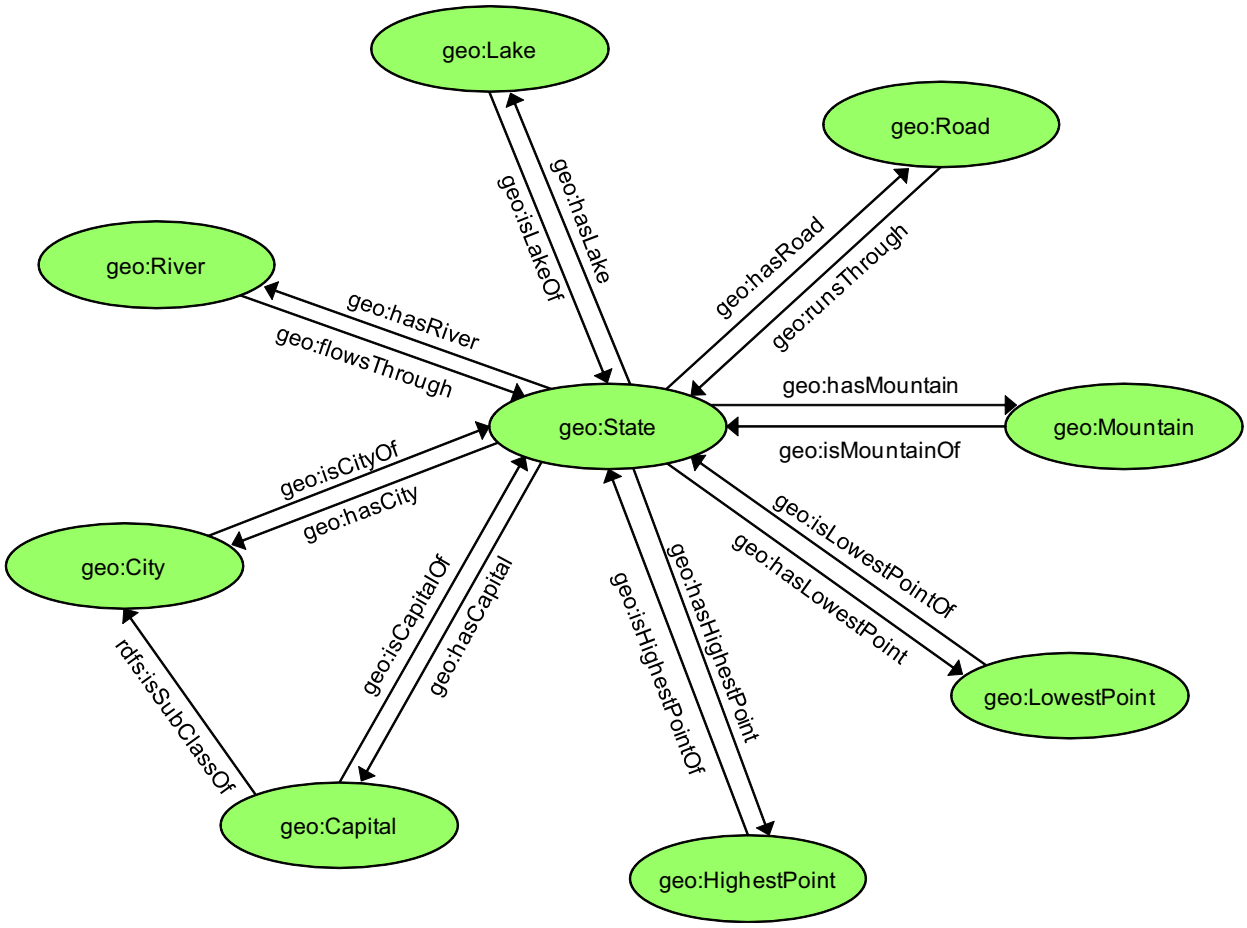


Figure A.1: The ontology model of the geography OWL knowledge base directly mapped from the original dataset specified as Prolog knowledge base.

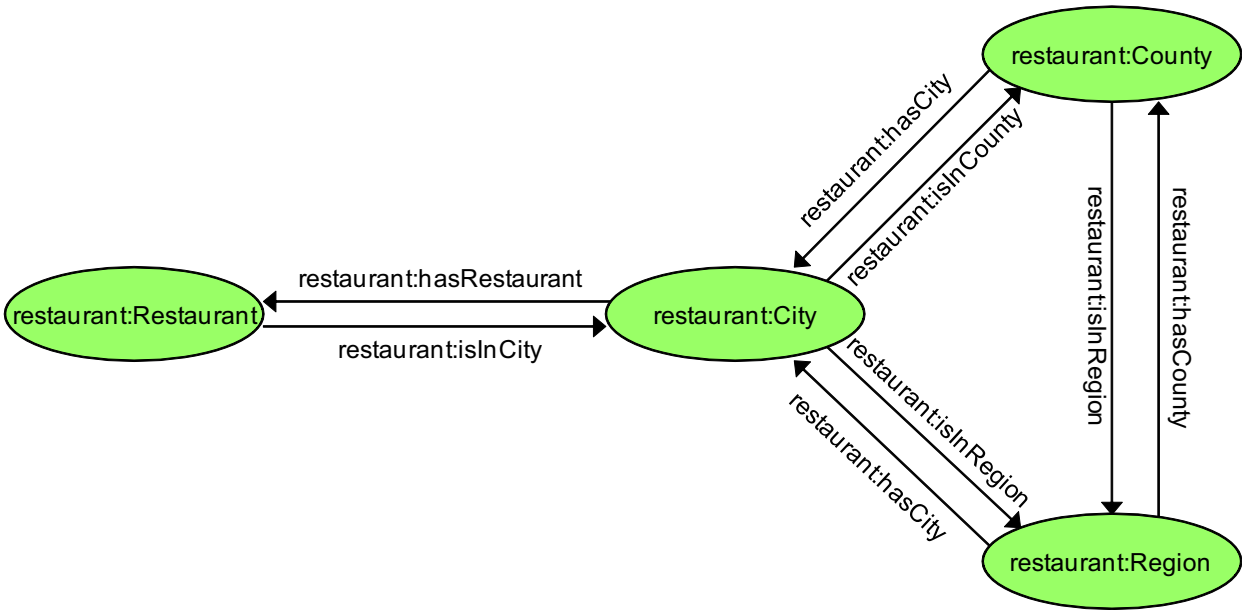


Figure A.2: The ontology model of the restaurant OWL knowledge base directly mapped from the original dataset specified as Prolog knowledge base.

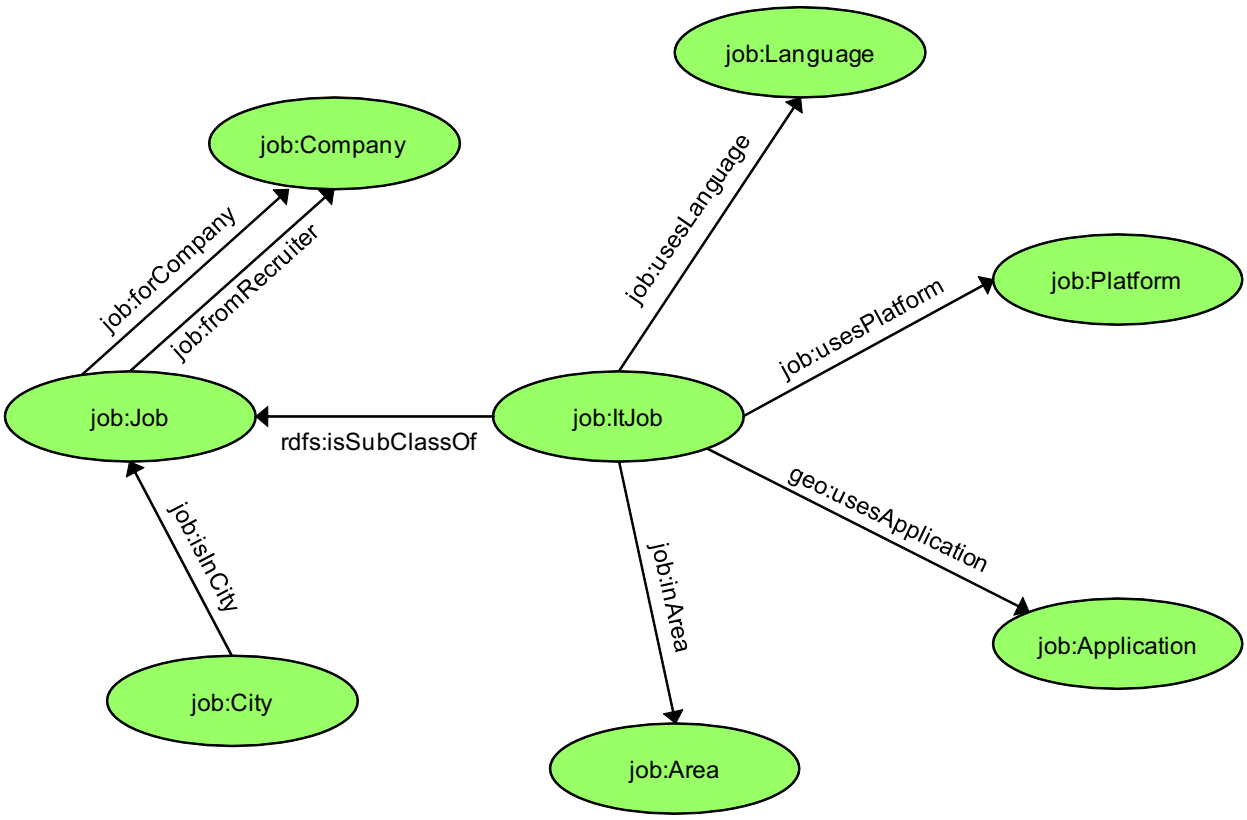


Figure A.3: The ontology model of the job postings OWL knowledge base directly mapped from the original dataset specified as Prolog knowledge base.

B

Appendix

The System Usability Scale (SUS) by John Brooke: SUS - A “quick and dirty” Usability Scale. In Jordan, P.W., Thomas, B. and Weerdmeester, B.A., and McClelland, A.L. (*eds.*): Usability Evaluation in Industry. Taylor and Francis, London 1996.

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Brooke, John: SUS - A ``quick and dirty" Usability Scale. In Jordan, P.W., Thomas, B. and Weerdmeester, B.A., and McClelland, A.L. (eds.): Usability Evaluation in Industry. Taylor and Francis, London 1996.

C

Appendix

The comparison questionnaire used in the usability study (Chapter 6), in which subjects were asked which of the four tested NLIs they liked best, which one they liked least and also which query language they liked best and which query language they liked least.

Fragebogen zum Vergleich der vier Suchsysteme

Kreuzen Sie bitte Zutreffendes an (bitte **nur 1 Kreuz** pro Frage) oder beantworten Sie die Fragen.

	1 NLP-Reduce	2 Ginseng	3 Querix	4 Semantic Crystal
				
1. Welches Suchsystem hat Ihnen am besten gefallen?	☐	☐	☐	☐
Weshalb?				
2. Welches Suchsystem hat Ihnen am wenigsten gefallen?	☐	☐	☐	☐
Weshalb?				
3. Die Abfragesprache von welchem System hat Ihnen am besten gefallen?	☐	☐	☐	☐
Weshalb?				
4. Die Abfragesprache von welchem System hat Ihnen am wenigsten gefallen?	☐	☐	☐	☐
8. Weshalb?				

☞ **Gehen Sie nun zur letzten Seite, um ein paar Angaben zur Ihrer Person zu machen.**

D

Appendix

The demographic questionnaire used in the usability study (Chapter 6) asking for age, gender, profession, knowledge of informatics, knowledge of linguistics, knowledge of formal query languages, and knowledge of English of the subjects.

Persönliche Angaben

Bitte geben Sie uns zum Abschluss noch die folgenden Angaben zu Ihrer Person.

Die Informationen werden für rein statistische Zwecke und nicht für individuelle Schlussfolgerungen benutzt.

1. Alter: ☐ Jahre

2. Geschlecht: ☐ weiblich ☐ männlich

3. Beruf: ☐

		professionell	gut	mittel	schlecht	gar keine
4.	Meine Kenntnisse in Linguistik (Sprachwissenschaft) sind:	☐	☐	☐	☐	☐
5.	Meine Kenntnisse in Informatik sind:	☐	☐	☐	☐	☐
6.	Meine Kenntnisse in Bezug auf formale Abfragesprachen (z.B. SQL, RDQL, SPARQL etc.) sind:	☐	☐	☐	☐	☐
7.	Meine Englischkenntnisse sind:	☐	☐	☐	☐	☐

Damit haben Sie es geschafft. Herzlichen Dank!

Sie können nun auf dem separaten Blatt Ihren Namen sowie Ihre Emailadresse angeben, falls Sie an den Ergebnissen der Untersuchung interessiert sind.

E

Appendix

The complete instructions and all questionnaires used in the usability study (Chapter 6) as presented to each subject, but with different order of the tools and queries for each subject.

University of Zurich
Department of Informatics

Dynamic and Distributed
Information Systems

Untersuchung zu natürlichsprachlichen Suchmaschinen

Wir danken Ihnen ganz herzlich dafür, dass Sie sich ca. **1 Stunde** Zeit nehmen und bei der Untersuchung mitmachen. Sie erhalten dafür nach dem Experiment **Fr. 25.-** in bar ausbezahlt.

Die vorliegende Untersuchung ist Teil eines Forschungsprojektes am Institut für Informatik der Universität Zürich, das **Abfrage- und Suchsysteme** entwickelt und evaluiert. Sie wird von der Forschungsgruppe *Dynamic and Distributed Information Systems* unter der Leitung von Prof. Abraham Bernstein und Esther Kaufmann durchgeführt.

Das Ziel der Untersuchung ist ein **Vergleich von verschiedenen Informationssuchsystemen**, die nach Antworten auf Fragen in semi-strukturierten Daten suchen. *Semi-strukturiert* heisst, dass die Daten nicht in einer Datenbank vorliegen, aber mit gewissen zusätzlichen Informationen versehen sind, sodass etwas mehr als nur die Daten selbst vorhanden sind. Im Experiment, welches Sie gleich durchspielen werden, interessiert in erster Linie die Benutzerfreundlichkeit der unterschiedlichen Abfragesprachen, mit denen Fragen an ein Suchsystem gestellt werden können.

Sämtliche Angaben werden selbstverständlich **streng vertraulich** behandelt, nur zu wissenschaftlichen Zwecken verwendet, verschlüsselt sowie passwortgeschützt gespeichert und nicht an Dritte weitergegeben.

Falls Sie Interesse an den Ergebnissen der Untersuchung haben, dann geben Sie bitte am Schluss der Fragenbeantwortung auf dem separat zur Verfügung stehenden Blatt Ihren Namen und Ihre E-Mail-Adresse an. Name und E-Mail-Adresse werden getrennt von Ihren Antworten aufbewahrt, sodass eine **anonyme Auswertung** der Untersuchungsresultate gewährleistet ist.

Um mit dem Experiment zu beginnen, gehen Sie bitte jetzt zur nächsten Seite.

Ihre Aufgaben

Sie werden nun **4 verschiedene Suchsysteme** testen, indem Sie an jedes System **4 Fragen**, die Ihnen präsentiert werden in der vom System verlangten Abfragesprache stellen. Sämtliche Instruktionen sind auf diesen Blättern vorhanden; die Fragen können Sie am Computer eingeben. Bei jedem Suchsystem lesen Sie zuerst eine Einführung in die Benutzung des Systems. Danach werden Ihnen stichwortartig formulierte Fragen in englischer Sprache präsentiert, und Sie formulieren diese Fragen in die Abfragesprache jedes Systems um. Folgen Sie einfach den Instruktionen. Sie brauchen die Antworten auf die Fragen **nicht** aufzuschreiben.

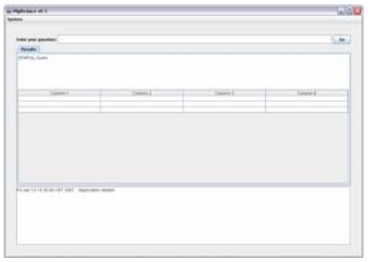
Am Schluss jedes Suchsystems füllen Sie einen **Fragebogen mit 10 Fragen** aus, mit welchem Sie die Nützlichkeit und die Benutzerfreundlichkeit des Systems und dessen Abfragesprache beurteilen können. Im Anschluss an das Testen der einzelnen Suchsysteme werden Sie eine **vergleichende Bewertung** vorzunehmen und uns dazu **8 Fragen** beantworten

Ganz am Schluss des Experiments bitten wir Sie noch, **7 Fragen** zur Ihrer Person zu beantworten.

Bitte beachten Sie, dass wir beim Experiment **nicht** Ihr Können testen, sondern dass Sie für uns die Suchsysteme beurteilen. Wir untersuchen **tatsächlich** die Benutzerfreundlichkeit der verschiedenen Suchsystem und nicht Ihr Verhalten.

Um mit dem ersten Suchsystem zu beginnen, gehen Sie bitte jetzt zur nächsten Seite.

Suchsystem 1: NLP-Reduce



Bitte lesen Sie jetzt diese Seite als Einführung zum Suchsystem 1.

Sie sehen auf dem Bildschirm vor Ihnen das Suchsystem NLP-Reduce, mit welchem man Fragen zur Geografie der USA stellen kann, die vom System beantwortet werden.

Beim diesem Suchsystem sind **vollständige englische Fragen** oder nur **Satzteile** oder nur **Stichwörter** als Eingaben erlaubt.

Das System erkennt **keine Superlative** (z.B. *biggest*, *largest*, *lowest* etc.)

Um eine Frage oder einen Satz einzugeben, klicken Sie mit der Maus in das weisse Textfeld nach „**Enter your question**“.

Jede Frage kann mit einem Fragezeichen abgeschlossen werden, muss aber nicht.

Gross- und Kleinschreibung muss nicht eingehalten werden.

Ist eine Frage vollständig eingegeben, kann durch Klicken auf „Go“ oder Drücken der Enter-/Eingabetaste die Beantwortung der Frage ausgelöst werden.

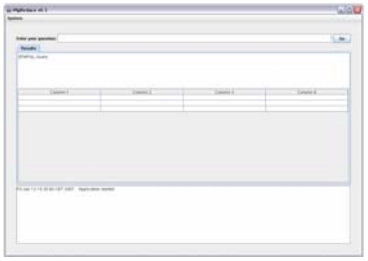
Sie können jede Frage ändern, indem Sie mit der Maus an die entsprechende Position der Frage klicken und den Text verändern. Durch erneutes Klicken auf „Go“ oder das Drücken der Enter-/Eingabetaste wird die Frage wiederum beantwortet.

Sie können auch jede Frage oder jeden Satz mit der Maus markieren und auf der Tastatur „Delete“ drücken, um die ganze Frage zu löschen, sodass Sie eine neue Frage eingeben können.

Die vom System gefundenen Antworten auf Ihre Fragen werden in Tabellenform angezeigt. Unten im Systemfenster wird die Meldung „**No hits found!**“ angeben, falls eine Frage nicht beantwortet werden kann.

☞ **Gehen Sie nun zur nächsten Seite, um mit dem Suchsystem 1 zu beginnen.**

Fragen an das Suchsystem 1: NLP-Reduce



Damit Sie sich mit dem Suchsystem 1 vertraut machen können, bitten wir Sie, zuerst eine Frage als Probedurchgang an das System zu stellen. Formulieren Sie dazu die folgende Frage in die Abfragesprache des Systems um, und lassen Sie die Frage vom System beantworten, indem Sie nach dem Eingeben der Frage auf „Go“ oder auf die Enter-/Eingabetaste klicken.

Nashville capital of state

Nun geht es an die Umformulierung der eigentlichen 4 Testfragen. Bitte formulieren Sie die nachfolgenden 4 Stichwort-Fragen nacheinander in der vom Suchsystem NLP-Reduce erlaubten Abfragesprache. Sie können jederzeit zur nächsten Frage weitergehen, auch wenn das System die Antwort zu einer Frage nicht gefunden hat.

1. **how many/number of rivers run through (state) Kansas?**

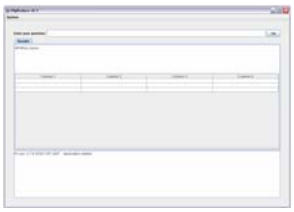
2. **rivers that run through state that has largest city in US?**

3. **size/area of (state) Alaska?**

4. **states that have city (named) Springfield?**

Wenn Sie die 4 Fragen umformuliert und eingegeben haben, füllen Sie nun bitte den Fragebogen auf der nächsten Seite aus.

Fragebogen zur Benutzerfreundlichkeit und Verwendbarkeit vom Suchsystem 1: NLP-Reduce



Kreuzen Sie bitte Zutreffendes an. Bitte **nur 1 Kreuz** pro Frage.

		vollkommen einverstanden	einverstanden	weder noch	nicht einverstanden	ganz und gar nicht einverstanden
1.	Ich denke, dass ich das System und dessen Abfragesprache gerne öfters benutzen würde.	☐	☐	☐	☐	☐
2.	Ich finde, dass das System und dessen Abfragesprache unnötig kompliziert waren.	☐	☐	☐	☐	☐
3.	Ich denke, dass das System und dessen Abfragesprache einfach zu benutzen waren.	☐	☐	☐	☐	☐
4.	Ich denke, dass ich die Hilfe einer Fachperson benötigen würde, um das System und dessen Abfragesprache benützen zu können.	☐	☐	☐	☐	☐
5.	Ich finde, dass die verschiedenen Aufgaben des Systems und der Abfragesprache gut integriert sind.	☐	☐	☐	☐	☐
6.	Ich denke, dass es zu viele Widersprüchlichkeiten in diesem System und der Abfragesprache gibt.	☐	☐	☐	☐	☐
7.	Ich könnte mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System und der Abfragesprache umzugehen.	☐	☐	☐	☐	☐
8.	Ich finde, dass das System und die Abfragesprache sehr mühsam waren.	☐	☐	☐	☐	☐
9.	Ich fühlte mich sehr selbstsicher bei der Benutzung des Systems und der Abfragesprache.	☐	☐	☐	☐	☐
10.	Ich musste viele Dinge lernen, bevor ich mit dem System und der Abfragesprache umgehen konnte.	☐	☐	☐	☐	☐

☞ **Um mit den Fragen an das Suchsystem 2 zu beginnen, gehen Sie bitte jetzt zur nächsten Seite.**

Suchsystem 2: Ginseng

Bitte lesen Sie jetzt diese Seite als Einführung zum Suchsystem 2.

Klicken Sie unten auf dem Bildschirm auf das Rechteck „**Ginseng v0.86**“.

Sie sehen nun auf dem Bildschirm vor Ihnen das Suchsystem Ginseng. Der Name steht für *Guided Input Natural Language Search Engine*. Dieses Suchsystem beantwortet Fragen zur Geografie der USA.

Bei diesem Suchsystem können **nur Fragen eingegeben werden, die vom System vorgeschlagen werden und die den Wörtern folgen, die in den aufgezeigten Listen stehen**. Die Listen ermöglichen Ihnen, ganze Sätze zu bilden. Wenn Sie im weissen Textfeld nach „Ask a question“ zu tippen beginnen, werden Ihnen die Listen mit Wörtern angezeigt, die Sie eingeben können. Tippen Sie beispielsweise ein „w“ ein, dann erscheinen alle mit „w“ beginnenden möglichen Wörter. Je mehr Buchstaben Sie pro Wort tippen, desto mehr öffnet sich die Liste mit möglichen Wörtern oder schränkt sich die Liste ein.

Auf der rechten Seite des Suchsystemfensters sehen Sie die Elemente der Geografiedaten dargestellt.

Sie können **ein Wort vollständig eintippen** oder **mit der Maus auf ein Wort in der Liste klicken** oder **mit der Pfeiltaste nach unten zu dem Wort gehen**, das Sie eingeben möchten, und die **Eingabetaste (Enter) drücken**.

Ist ein Wort eingegeben, werden Ihnen die darauffolgenden erlaubten Wörter angezeigt. Sie können erneut ein Wort auswählen. Das System führt Sie so durch die Formulierung einer Frage hindurch.

Taucht ein **Fragezeichen** in der Liste auf, so kann damit ein Satz abgeschlossen werden. Nach dem Auswählen des Fragezeichens wird eine Frage vom System beantwortet und die Antworten im unteren weissen Feld angezeigt.

Mit der Zurück-Taste (*Backspace*) können Sie im Satz zurückgehen und Wörter wieder löschen, um den Satz beispielsweise anders weiterzufahren.

Um mit den Fragen an das Suchsystem 2 zu beginnen, gehen Sie bitte jetzt zur nächsten Seite.

Fragen an das Suchsystem 2: Ginseng

Damit Sie sich mit dem Suchsystem Ginseng vertraut machen können, bitten wir Sie, zuerst eine Frage als Probedurchgang an das System zu stellen. Formulieren Sie dazu die folgende Frage in die Abfragesprache des Systems um, und lassen Sie die Frage vom System beantworten, indem Sie am Schluss Ihrer Frage das Fragezeichen auswählen.

What is the capital of Arizona?

Nun geht es an die Umformulierung der eigentlichen 4 Testfragen. Bitte formulieren Sie die nachfolgenden 4 Stichwort-Fragen nacheinander in der vom Suchsystem Ginseng erlaubten Abfragesprache, d.h. mit den Wörtern, die in den Listen vorkommen. Sie können jederzeit zur nächsten Frage weitergehen, auch wenn das System die Antwort zu einer Frage nicht gefunden hat.

Wichtig: Bitte drücken Sie immer die **Clear-Taste**, bevor Sie eine neue Frage eingeben, damit die letzte Frage vollständig gelöscht wird.

1. **size/area of (state) Georgia?**

2. **states that have city (named) Arlington?**

3. **how many/number of rivers run through (state) Colorado?**

4. **rivers that run through state that has smallest city in US?**

Wenn Sie die 4 Fragen umformuliert und eingegeben haben, füllen Sie nun bitte den Fragebogen auf der nächsten Seite aus.

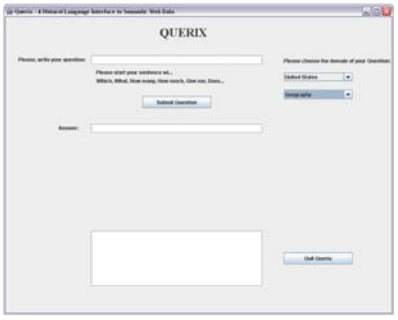
Fragebogen zur Benutzerfreundlichkeit und Verwendbarkeit vom Suchsystem 2: Ginseng

Kreuzen Sie bitte Zutreffendes an. Bitte **nur 1 Kreuz** pro Frage.

		vollkommen einverstanden	einverstanden	weder noch	nicht einverstanden	ganz und gar nicht einverstanden
1.	Ich denke, dass ich das System und dessen Abfragesprache gerne öfters benutzen würde.	☐	☐	☐	☐	☐
2.	Ich finde, dass das System und dessen Abfragesprache unnötig kompliziert waren.	☐	☐	☐	☐	☐
3.	Ich denke, dass das System und dessen Abfragesprache einfach zu benutzen waren.	☐	☐	☐	☐	☐
4.	Ich denke, dass ich die Hilfe einer Fachperson benötigen würde, um das System und dessen Abfragesprache benützen zu können.	☐	☐	☐	☐	☐
5.	Ich finde, dass die verschiedenen Aufgaben des Systems und der Abfragesprache gut integriert sind.	☐	☐	☐	☐	☐
6.	Ich denke, dass es zu viele Widersprüchlichkeiten in diesem System und der Abfragesprache gibt.	☐	☐	☐	☐	☐
7.	Ich könnte mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System und der Abfragesprache umzugehen.	☐	☐	☐	☐	☐
8.	Ich finde, dass das System und die Abfragesprache sehr mühsam waren.	☐	☐	☐	☐	☐
9.	Ich fühlte mich sehr selbstsicher bei der Benutzung des Systems und der Abfragesprache.	☐	☐	☐	☐	☐
10.	Ich musste viele Dinge lernen, bevor ich mit dem System und der Abfragesprache umgehen konnte.	☐	☐	☐	☐	☐

☞ **Gehen Sie nun zur nächsten Seite, um mit dem Suchsystem 3 zu beginnen.**

Suchsystem 3: Querix



Bitte lesen Sie jetzt diese Seite als Einführung zum Suchsystem 3.

Klicken Sie unten auf dem Bildschirm auf das Rechteck „**Querix – A Natural**“.

Sie sehen auf dem Bildschirm vor Ihnen das Suchsystem Querix, mit welchem man Fragen zur Geografie der USA stellen kann. Das System liefert dann die Antworten auf die Fragen.

Beim diesem Suchsystem werden die Fragen oder Befehlssätze in Form von **ganzen englischen Sätzen** eingegeben. Erlaubt sind nur Fragen oder Sätze, die folgendermassen beginnen:

Which … ?
What … ?
How many …?
How much … ?
Does … ?
Give me … .

Um eine Frage oder einen Satz einzugeben, klicken Sie mit der Maus in das weisse Textfeld nach „Please, write your question“.

Jede Frage wird mit einem **Fragezeichen** abgeschlossen, jeder Befehlssatz mit einem **Punkt**.

Gross- und Kleinschreibung muss nicht eingehalten werden.

Ist eine Frage/ein Satz vollständig eingegeben, kann durch Klicken auf „**Submit Question**“ die Beantwortung der Frage ausgelöst werden.

Entdeckt das System eine Unklarheit in einer Frage, so öffnet sich ein Fenster „**AskBox**“, in welchem Sie die beabsichtigte Bedeutung der Frage auswählen können, um so die Frage für das System zu präzisieren. Klicken Sie dazu auf die beabsichtigte Bedeutung und dann auf „**Submit**“.

Sie können jede Frage ändern, indem Sie mit der Maus an die entsprechende Position der Frage klicken und den Text verändern. Durch erneutes Klicken auf „**Submit Question**“ wird die Frage wiederum beantwortet.

Sie können auch jede Frage oder jeden Satz mit der Maus markieren und auf der Tastatur „Delete“ (löschen) drücken, um die ganze Frage zu löschen, sodass Sie eine neue Frage eingeben können.

Die vom System gefundenen Antworten auf Ihre Fragen werden bei „**Answer**“ angezeigt. Unten im Systemfenster wird die formale Repräsentation Ihrer in Englisch formulierten Frage angezeigt.

☞ **Um mit den Fragen an das Suchsystem 3 zu beginnen, gehen Sie bitte jetzt zur nächsten Seite.**

Fragen an das Suchsystem 3: Querix

Damit Sie sich mit dem Suchsystem 3 vertraut machen können, bitten wir Sie, zuerst eine Frage als Probedurchgang an das System zu stellen. Formulieren Sie dazu die folgende Frage in die Abfragesprache des Systems um, und lassen Sie die Frage vom System beantworten, indem Sie nach dem Eingeben der Frage auf „Submit Question“ klicken.

What is the capital of Montana?

Nun geht es an die Umformulierung der eigentlichen 4 Testfragen. Bitte formulieren Sie die nachfolgenden 4 Stichwort-Fragen nacheinander in der vom Suchsystem Querix erlaubten Abfragesprache und lassen Sie sich die Frage vom System beantworten. Sie können jederzeit zur nächsten Frage weitergehen, auch wenn das System die Antwort zu einer Frage nicht gefunden hat.

1. **rivers that run through state that has largest city in US?**

2. **how many/number of lakes in (state) California?**

3. **states that have city Portland?**

4. **size/area of (state) Florida?**

Wenn Sie die 4 Fragen umformuliert und eingegeben haben, füllen Sie nun bitte den Fragebogen auf der nächsten Seite aus.

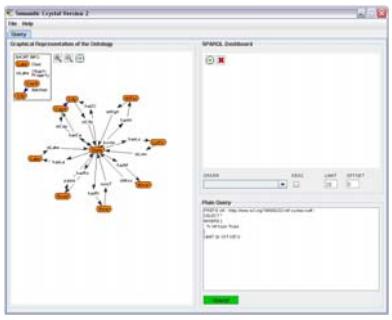
Fragebogen zur Benutzerfreundlichkeit und Verwendbarkeit vom Suchsystem 3: Querix

Kreuzen Sie bitte Zutreffendes an. Bitte **nur 1 Kreuz** pro Frage.

		vollkommen einverstanden	einverstanden	weder noch	nicht einverstanden	ganz und gar nicht einverstanden
1.	Ich denke, dass ich das System und dessen Abfragesprache gerne öfters benutzen würde.	☐	☐	☐	☐	☐
2.	Ich finde, dass das System und dessen Abfragesprache unnötig kompliziert waren.	☐	☐	☐	☐	☐
3.	Ich denke, dass das System und dessen Abfragesprache einfach zu benutzen waren.	☐	☐	☐	☐	☐
4.	Ich denke, dass ich die Hilfe einer Fachperson benötigen würde, um das System und dessen Abfragesprache benützen zu können.	☐	☐	☐	☐	☐
5.	Ich finde, dass die verschiedenen Aufgaben des Systems und der Abfragesprache gut integriert sind.	☐	☐	☐	☐	☐
6.	Ich denke, dass es zu viele Widersprüchlichkeiten in diesem System und der Abfragesprache gibt.	☐	☐	☐	☐	☐
7.	Ich könnte mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System und der Abfragesprache umzugehen.	☐	☐	☐	☐	☐
8.	Ich finde, dass das System und die Abfragesprache sehr mühsam waren.	☐	☐	☐	☐	☐
9.	Ich fühlte mich sehr selbstsicher bei der Benutzung des Systems und der Abfragesprache.	☐	☐	☐	☐	☐
10.	Ich musste viele Dinge lernen, bevor ich mit dem System und der Abfragesprache umgehen konnte.	☐	☐	☐	☐	☐

☞ **Gehen Sie nun zur nächsten Seite, um mit dem Suchsystem 4 zu beginnen.**

Suchsystem 4: Semantic Crystal



Bitte lesen Sie jetzt diese zwei Seiten als Einführung zum Suchsystem 4.

Klicken Sie unten auf dem Bildschirm auf das Rechteck „**Semantic Crystal**“.

Sie sehen nun auf dem Bildschirm vor Ihnen das Suchsystem Semantic Crystal, mit welchem man Fragen zur Geografie der USA stellen kann. Das System liefert dann die Antworten auf die Fragen.

Beim diesem Suchsystem werden die Fragen nicht sprachlich, sondern **mithilfe einer grafischen Darstellung** und **durch Klicken auf die Elemente** in der grafischen Darstellung eingegeben.

Auf der linken Seite im Fenster „Graphical Representation of the Ontology“ sehen Sie die grafische Darstellung der Geografiedaten. Grundsätzlich werden **orangefarbene Elemente** durch Klicken darauf ausgewählt und über **blaufarbene Eigenschaften** mit anderen orangefarbenen Elementen verknüpft. Zusätzlich werden **Restriktionen in Form von Elementnamen** sowie das, was das System als Antwort auf eine Frage zurückliefern soll, angegeben.

Um Ihnen das System vorzuführen, wird nun die Umformulierung der Frage „Frankfort is the capital of what state?” Schritt für Schritt vorgeführt. Bitte machen Sie jeden Schritt am Bildschirm nach, damit Sie sich mit dem System vertraut machen können.

Schritt 1
In der Frage „Frankfort capital of what state?” geht es um eine bestimmte Hauptstadt (*capital*). Somit klicken wir auf **das orange Element „Capital“** auf der linken Seite des Fensters. Eine Auswahlliste erscheint, wo wir auswählen können, welche Eigenschaft diese Hauptstadt haben soll. Wir wählen „**isCapitalOf**“ aus, da wir wissen wollen, von welchem Staat (*state*) Frankfort die Hauptstadt ist. Im rechten Fenster „SPARQL Dashboard“ erscheint nun eine Darstellung eines Elements „**Capital**“, das eine Verbindung „**isCapitalOf**“ zum Element „**State**“ hat.

Schritt 2
Wir müssen nun definieren, dass die Hauptstadt den Namen „Frankfort“ hat. Dazu klicken wir auf der linken Seite des Fensters auf das Element „**Capital**“. Eine Auswahlliste wird geöffnet. Wir fahren in dieser Auswahlliste mit dem Mauszeiger auf „**rdfs:label**“, denn der Name eines Elements wird als „label“ bezeichnet. Wir erhalten dann eine weitere Auswahl, wo wir auf „**Restriction**“ klicken, um dem Element „Capital“ eben die Restriktion des Namens „Frankfort“ zu geben. Im Dashboard auf der rechten Seite erscheint ein grünes Element „**SET RESTR!**“, das eine Verbindung zu unserem Element „**Capital**“ hat.

Schritt 3
Wenn wir nun **auf dieses grüne Element im Dashboard links klicken** und dann in der sich öffnenden Auswahl auf „**Change Restriction Value**“ klicken, öffnet sich ein weisses Textfeld, wo wir einen Namen reinchreiben können. In unserem Fall schreiben wir „Frankfort“ und drücken dann auf „Ok“. Sie sehen nun, dass die Beschriftung des grünen Elements „Frankfort“ heisst. Gross- und Kleinschreibung spielt in den Textfeldern keine Rolle.

Schritt 4
Dies ist ein wichtiger Schritt. Denn wir müssen noch deklarieren, was vom Suchsystem denn nach dem Finden einer Antwort überhaupt ausgegeben werden soll, da dies nicht automatisch geschieht. Aufgrund unserer Frage „Frankfort capital of what state?" sind wir auf der Suche nach dem Staat, der Frankfort als Hauptstadt hat. Wir klicken also im linken Fenster auf das gesuchte Element „state", gehen in der Auswahl mit dem Mauszeiger auch hier auf „**rdfs:label**", weil wir ja den Namen des gesuchten Staates wissen wollen. Dieses Mal aber klicken wir auf „**Output**" (und nicht auf „Restriction"), **sodass das System erkennt, dass dieser Name des Staates ausgegeben werden soll**. Im Dashboard erscheint der definierte Output als gelbes Element. Falls es mehrere Staaten mit einer Hauptstadt Frankfort geben sollte, werden automatisch alle Namen dieser Staaten ausgegeben.

Unsere Frage ist jetzt vollständig.

Schritt 5
Nun können wir auf den grünen Knopf „**Query!**" unten rechts im Fenster klicken, um die Frage vom System beantworten zu lassen. Unten links im Systemfenster wird die formale Repräsentation Ihrer zusammengestellen Frage angezeigt.

Nach Klicken auf „**Query!**" wird eine Liste mit dem Resultat unserer Anfrage angezeigt. Der gesuchte Staat, dessen Hauptstadt Frankfort heisst, ist Kentucky.

Um wieder zurück zum Fragezusammenstellungsfenster zu gelangen, klicken Sie oben links auf „**Query**". Sie können im Dashboard die Frage nach Belieben verändern und wiederum beantworten lassen. Wenn Sie die vorhergehende Frage im Dashboard löschen möchten, um eine ganz andere Frage zusammenzustellen, so können Sie **im Dashboard** auf das rote „**X**" neben dem grünen Pfeil klicken. Falls Sie die Frage wirklich löschen wollen, bestätigen Sie das durch Klicken auf „Ja".

Und hier noch ein paar weitere Möglichkeiten und Tipps beim Zusammenstellen von Fragen:

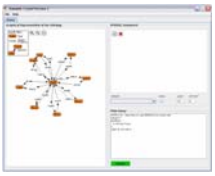
Indem Sie **ein oranges Element im linken Fenster anklicken** und in der Auswahl „**Add class token**" wählen, können Sie dieses Element ihrer Frage im Dashboard hinzufügen. Durch Klicken auf das Element im Dashboard und durch Auswählen einer passenden Eigenschaft wird das Element automatisch mit den bisherigen Elementen im Dashboard verknüpft.

Wenn Sie im Dashboard z.B. bereits einen Fluss (*river*) und eine Eigenschaft „**runsThrough**" (*fliesst durch*) haben, die mit dem Element „**State**" verbunden ist, so können Sie beispielsweise auch dem Staat noch Eigenschaften hinzufügen. Dazu klicken Sie in der grafischen Darstellung links auf das Element „**State**" und wählen eine Eigenschaft des Staates aus, z.B. „**hasCapital**". **Auf diese Weise wird die Eigenschaft des Staates automatisch mit dem Staat verbunden, der bereits im Dashboard ist**. Das ermöglicht Ihnen, kompliziertere Fragen mit mehreren verbundenen Elementen zusammenzustellen, also zum Beipspiel: „Which river runs through a state that has the capital Frankfort?"

Tipp: Wenn Sie mit dem Mauszeiger **auf ein Element oder einen Aktionsknopf fahren**, so wird Ihnen automatisch angezeigt, wofür der Knopf ist oder was die vollständige Beschriftung eines Elements ist.

Um mit den Fragen an das Suchsystem 4 zu beginnen, gehen Sie bitte jetzt zur nächsten Seite.

Fragen an das Suchsystem 4: Semantic Crystal



Nun geht es an die Umformulierung der 4 Testfragen. Bitte formulieren Sie die nachfolgenden 4 Stichwort-Fragen nacheinander in der vom Suchsystem Semantic Crystal erlaubten Abfragesprache, und lassen Sie die Fragen vom System beant-worten, indem Sie nach dem Eingeben jeder Frage auf den grünen Knopf „**Query!**“ klicken. Sie können jederzeit zur nächs-en Frage weitergehen, auch wenn das System die Antwort zu einer Frage nicht gefunden hat.

Sie finden in der untenstehenden Tabelle noch einmal die wichtigsten Bedienungstipps, an die Sie sich halten können. Bitte halten Sie sich vor allem an die rot markierten Instruktionen.

1. **states that have city (named) Rochester?**

2. **size/area of (state) Hawaii?**

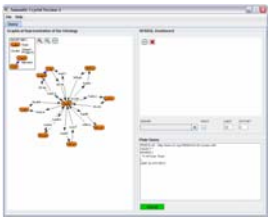
3. **rivers that run through state that has smallest city in US?**

4. **how many/number of lakes in (state) Wisconsin?**

Eigenschaften eines Elements **auflisten** lassen	• auf oranges Element **im linken Fenster** klicken
Element und Eigenschaft ins Dashboard einfügen	• auf oranges Element **im linken Fenster** klicken • eine passende Eigenschaft auswählen und anklicken
zusätzliches Element dem Dashboard hinzufügen	• auf oranges Element **im linken Fenster** klicken • eine Eigenschaft auswählen • das Element und die Eigenschaft werden automatisch mit den Elementen im Dashboard verknüpft
zusätzliche Eigenschaft einem Element im Dashboard hinzufügen	• auf das orange Element **im linken Fenster** klicken • eine Eigenschaft auswählen • das Element und die Eigenschaft werden automatisch mit den Elementen im Dashboard verknüpft
einen bestimmten **Namen** eingeben	• auf das entsprechende orange Element **im linken Fenster** klicken • rdfs:label auswählen • „Restriction“ auswählen • auf „**SET RESTR!**“ im Dashboard klicken • „Change Restriction Value“ auswählen • Wort/Namen eingeben und „Ok“ drücken
festlegen, was das System als Antwort **ausgeben** soll (sonst wird das Resultat nicht angezeigt)	• auf das entsprechende orange Element **im linken Fenster** klicken • rdfs:label auswählen • „Output“ wählen
Frage **beantworten** lassen	• auf den grünen Knopf „**Query!**“ klicken
von der Antwortseite zur grafischen Repräsentation **wechseln**	• auf „Query“ oben links klicken
Elemente im Dashboard **neu anordnen** lassen	• auf **den grünen Pfeil** im Dashboard klicken
Dashboard **löschen**	• auf das rote **X** im Dashboard und dann auf „Ja“ klicken

Wenn Sie die 4 Fragen umformuliert und eingegeben haben, füllen Sie nun bitte den Fragebogen auf der nächsten Seite aus.

Fragebogen zur Benutzerfreundlichkeit und Verwendbarkeit vom Suchsystem 4: Semantic Crystal



Kreuzen Sie bitte Zutreffendes an. Bitte **nur 1 Kreuz** pro Frage.

		vollkommen einverstanden	einverstanden	weder noch	nicht einverstanden	ganz und gar nicht einverstanden
1.	Ich denke, dass ich das System und dessen Abfragesprache gerne öfters benutzen würde.	☐	☐	☐	☐	☐
2.	Ich finde, dass das System und dessen Abfragesprache unnötig kompliziert waren.	☐	☐	☐	☐	☐
3.	Ich denke, dass das System und dessen Abfragesprache einfach zu benutzen waren.	☐	☐	☐	☐	☐
4.	Ich denke, dass ich die Hilfe einer Fachperson benötigen würde, um das System und dessen Abfragesprache benützen zu können.	☐	☐	☐	☐	☐
5.	Ich finde, dass die verschiedenen Aufgaben des Systems und der Abfragesprache gut integriert sind.	☐	☐	☐	☐	☐
6.	Ich denke, dass es zu viele Widersprüchlichkeiten in diesem System und der Abfragesprache gibt.	☐	☐	☐	☐	☐
7.	Ich könnte mir vorstellen, dass die meisten Leute sehr schnell lernen würden, mit dem System und der Abfragesprache umzugehen.	☐	☐	☐	☐	☐
8.	Ich finde, dass das System und die Abfragesprache sehr mühsam waren.	☐	☐	☐	☐	☐
9.	Ich fühlte mich sehr selbstsicher bei der Benutzung des Systems und der Abfragesprache.	☐	☐	☐	☐	☐
10.	Ich musste viele Dinge lernen, bevor ich mit dem System und der Abfragesprache umgehen konnte.	☐	☐	☐	☐	☐

☞ **Gehen Sie nun zur nächsten Seite, um den Fragebogen zum Vergleich der vier Suchsysteme auszufüllen.**

Fragebogen zum Vergleich der vier Suchsysteme

Kreuzen Sie bitte Zutreffendes an (bitte **nur 1 Kreuz** pro Frage) oder beantworten Sie die Fragen.

	1 NLP-Reduce	2 Ginseng	3 Querix	4 Semantic Crystal
				
1. Welches Suchsystem hat Ihnen am besten gefallen?	☐	☐	☐	☐
Weshalb?				
2. Welches Suchsystem hat Ihnen am wenigsten gefallen?	☐	☐	☐	☐
Weshalb?				
3. Die Abfragesprache von welchem System hat Ihnen am besten gefallen?	☐	☐	☐	☐
Weshalb?				
4. Die Abfragesprache von welchem System hat Ihnen am wenigsten gefallen?	☐	☐	☐	☐
8. Weshalb?				

☞ **Gehen Sie nun zur letzten Seite, um ein paar Angaben zur Ihrer Person zu machen.**

Persönliche Angaben

Bitte geben Sie uns zum Abschluss noch die folgenden Angaben zu Ihrer Person.

Die Informationen werden für rein statistische Zwecke und nicht für individuelle Schlussfolgerungen benutzt.

1. Alter: ☐ Jahre

2. Geschlecht: ☐ weiblich ☐ männlich

3. Beruf: ☐

		professionell	gut	mittel	schlecht	gar keine
4.	Meine Kenntnisse in Linguistik (Sprachwissenschaft) sind:	☐	☐	☐	☐	☐
5.	Meine Kenntnisse in Informatik sind:	☐	☐	☐	☐	☐
6.	Meine Kenntnisse in Bezug auf formale Abfragesprachen (z.B. SQL, RDQL, SPARQL etc.) sind:	☐	☐	☐	☐	☐
7.	Meine Englischkenntnisse sind:	☐	☐	☐	☐	☐

Damit haben Sie es geschafft. Herzlichen Dank!

Sie können nun auf dem separaten Blatt Ihren Namen sowie Ihre Emailadresse angeben, falls Sie an den Ergebnissen der Untersuchung interessiert sind.

Sie haben Interesse an den Ergebnissen der Untersuchung?

Dann geben Sie doch hier Vorname, Name und Ihre Emailadresse an.

Um die Untersuchungsergebnisse **anonym** auswerten und archivieren zu können, werden wir diese Angaben getrennt von den restlichen Angaben aufbewahren.

Vorname:	
Name:	
Email:	

Curriculum Vitae

Personal Information

Name:	Esther Kaufmann
Nationality:	Swiss
Place of Citizenship:	Schaenis-Ruettiberg SG

Education

Doctoral Study Program
Department of Informatics, Faculty of Economics, Business Administration and Information Technology, University of Zurich
Main subject: Informatics
Minor subject: Information Management
Degree: Dr. Inform. / Ph.D.

Master Study Program
Faculty of Arts, University of Zurich
Main subject: German Linguistics and Literature
First minor subject: Computational Linguistics
Second minor subject: English Linguistics
Degree: Licentiata Philosophiae / Master of Arts

Grammar / High School
Cantonal School in Sargans SG
Degree: Matura Type E (Economics)